TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Professur Softwaretechnik

Bachelorarbeit

Erstellung einer einheitlichen Taxonomie für die Programmiermodelle der parallelen Programmierung

Markus Nestmann

Chemnitz, den 14. November 2016

Prüfer: Prof. Dr.-Ing. Steffen Becker
Betreuer: Dr.-Ing. Marcus Hilbrich

Danksagung

Zunächst möchte ich mich bei all denen bedanken, die während der Anfertigung dieser Arbeit an mich gedacht und mich motiviert haben.

Vielen Dank auch an meinen Betreuer, Herrn Dr.-Ing. Marcus Hilbrich, für die stetige Unterstützung. Er konnte mir bei meinen Fragen und meinen Untersuchungen durch sein Wissen über die Thematik dieser Arbeit und seine konstruktive Kritik immer wieder weiterhelfen.

Auch will ich mich bei meinen Eltern bedanken, die mich immer unterstützten. Vielen Dank an meine Mutter, die in genauer und zeitaufwendiger Weise die Arbeit durchlas und viele meiner Rechtschreibfehler ausfindig machen konnte.

Abstrakt

Durch die parallele Programmierung wird ermöglicht, dass Programme nebenläufig auf mehreren CPU-Kernen oder CPUs ausgeführt werden. Um das parallele Programmieren zu erleichtern, wurden diverse Sprachen (z.B. Erlang) und Bibliotheken (z.B. OpenMP) aufbauend auf parallele Programmiermodelle (z.B. Parallel Random Access Machine) entwickelt. Möchte z.B. ein Softwarearchitekt sich in einem Projekt für ein Programmiermodell entscheiden, muss er dabei auf mehrere wichtige Kriterien (z.B. Abhängigkeiten zur Hardware) achten. Erleichternd für diese Suche sind Übersichten, die die Programmiermodelle in diesen Kriterien unterscheiden und ordnen. Werden die existierenden Übersichten jedoch betrachtet, finden sich Unterschiede in der Klassifizierung, den verwendeten Begriffen und den aufgeführten Programmiermodellen. Diese Arbeit begleicht dieses Defizit, indem zuerst durch ein Systematic Literature Review die existierenden Taxonomien gesammelt und analysiert werden. Darauf aufbauend wird eine einheitliche Taxonomie erstellt. Mit dieser Taxonomie kann eine Übersicht über die parallelen Programmiermodelle erstellt werden. Diese Übersicht wird zusätzlich durch Informationen zu den jeweiligen Abhängigkeiten der Programmiermodelle zu der Hardware-Architektur erweitert werden. Der Softwarearchitekt (oder Projektleiter, Softwareentwickler,...) kann damit eine informierte Entscheidung treffen und ist nicht gezwungen alle Programmiermodelle einzeln zu analysieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Implementierung ..eines Programmiermodells; Ist beispielsweise eine erweiternde Bibliothek für Programmiersprachen oder eine eigene Programmiersprache. Setzt mindestens ein Programmiermodell um. Bsp.: Erlang [Erl], Cilk [Cil].

Konzept Das grundlegende Prinzip der Implementierung eines Programmiermodells. Das heißt: Welche Programmiermodelle werden genutzt? Wie werden diese eingesetzt? Bsp.: Erlang ist eine Programmiersprache (Implementierung), die das Aktorenmodell (Programmiermodell) als Konzept benutzt. Ihr Fokus liegt auf ressourcenschonend implementierten Prozessen.

Programmiermodell Eine abstrahierende Abbildung der Architektur von Parallelrechnern. Beinhaltet verschiedene Technologien, sowie Anforderungen an Hardware und Software. Bsp.: actor model [Agh85], circuit.

1. Einleitung

1.1. Einordnung des Themas

Diese Arbeit beschäftigt sich mit den Programmiermodellen aus der parallelen Programmierung. In der parallelen Programmierung werden mehrere parallele CPU-Kerne benutzt, um die Laufzeit von Programmen zu verkürzen. Diese Methode wurde betrachtet und genutzt, da die bis dahin stetige Entwicklung von schnelleren CPU-Kernen an ihre Grenzen stieß. Eine Erhöhung der Taktrate eines CPU-Kerns führte z.B. unter anderem zu Hitzeproblemen. Die stetige Herstellung schnellerer CPUs wird nun durch die Benutzung mehrerer paralleler CPU-Kerne fortgesetzt. Auf diesen Kernen kann ein Programm nebenläufig ausgeführt werden. Das Programm muss jedoch für diese Abarbeitung speziell aufgebaut sein, damit die einzelnen Programmabschnitte effektiv parallel bearbeitet werden können. Verschiedene Möglichkeiten wurden geschaffen, um parallel zu programmieren: Eigenständige parallele Programmiersprachen (z.B. Erlang) oder erweiternde Bibliotheken (z.B. OpenMP). Diese Möglichkeiten weisen unterschiedliche zugrundeliegende Konzepte auf. Die hier eingesetzten Programmiermodelle bilden, wie schon erwähnt, die Grundlage dieser Arbeit.

1.2. Motivation

Ein Blick auf existierende Übersichten zu den Programmiermodellen lässt mehrere Defizite erkennen: Die Übersichten enthalten nicht alle Programmiermodelle, sie unterscheiden nicht ausreichend zwischen den Programmiermodellen und sind untereinander inkonsistent. Diese Inkonsistenzen zeigen, dass zurzeit keine einheitliche Taxonomie für die Programmiermodelle der parallelen Programmierung existiert.

Nun entstehen dadurch aber verschiedene Probleme, wie folgende Beispiele zeigen. Es tritt z.B. ein Mangel an Informationen auf, da keine vollständige und einheitliche Übersicht gefunden werden kann. Für Softwareprojekte in diesem Bereich sind diese fehlenden Informationen jedoch notwendig, denn es werden dort Programmiermodelle benutzt. Hier sollen natürlich informierte Entscheidungen getroffen werden. Dies zeigt, dass es nötig ist, dem entgegenzuwirken und eine einheitliche Taxonomie zu erstellen. Folgende vier Punkte sollen das weiter hervorheben.:

- Die Leitung eines Softwareprojekts in dem Bereich der parallelen Programmierung will sich für ein Programmiermodell entscheiden. Nun ist es interessant zu wissen, welches Programmiermodell zu der benutzen Hardware in dem Projekt passt. Oder im Gegensatz dazu: Welche Hardware gekauft werden muss, um optimal an das Programmiermodell angepasst zu sein. Wichtige Gesichtspunkte

sind z.B. geteilter/gemeinsamer Speicher. Aufgrund der fehlenden einheitlichen Übersicht, müssen die Programmiermodelle einzeln analysiert werden. Es soll deswegen in dieser Arbeit herausgefunden werden, welche Hardwarekomponenten für die einzelnen Programmiermodelle interessant sind.

- Es werden immer wieder neue Implementierungen entwickelt. Muss sich ein Programmierer nun mit ständig neuen Konzepten befassen? Dies erübrigt sich, wenn sich diese nur z.B. in dem Design der GUI unterscheiden. Um diese Fragen zu beantworten, müssen die Unterschiede zwischen den Programmiermodellen bekannt sein. Deswegen werden in dieser Arbeit die einzelnen Implementierungen bezüglich ihrer Konzepte verglichen. Dies geschieht aufbauend auf den Informationen in den gefundenen Übersichten.

- Ein weiterer Programmierer möchte ein Programmiermodell für eine eigene Implementierung auswählen. Für ihn ist es wichtig zu erfahren, ob die Kurzübersichten (siehe 1.1) ausreichend sind. Um sich für ein Programmiermodell zu entscheiden, sollten die Alternativen bekannt sein. Das am besten passende Programmiermodell kann sonst übersehen werden. Sollten also die vorherrschenden Übersichten nicht ausreichend sein, müssen die darin ausgelassenen Programmiermodelle hinzugefügt werden.

- In wissenschaftlichen Arbeiten in der parallelen Programmierung finden sich z.B. Begriffe für die Klassifizierung von Programmiermodellen. Der Vergleich von existierenden Taxonomien zeigt jedoch, dass der selbe Begriff in der einen Taxonomie etwas Unterschiedliches in der anderen bezeichnen kann. Es kommt auch vor, dass unterschiedliche Begriffe dasselbe bezeichnen. Da das Ziel der Arbeit eine einheitliche Taxonomie ist, müssen natürlich auch die einzelnen Bestandteile aussagekräftig bezeichnet werden.

In der Tabelle 1.1, konnten einige Defizite der Übersichten sichtbar gemacht werden. Diese Tabelle wurde gewählt, da es sich um den Startpunkt der Arbeit handelt. Sie befindet sich in dem Wikipedia-Eintrag zu parallelen Programmiermodellen [Wik16]. Wird in diesem Gebiet gesucht, kann diese Übersicht somit schnell gefunden werden. Folgende Defizite existieren:

Die Tabelle enthält erstens nicht alle Programmiermodelle. Das Speichermodell PGAS ist z.B. nicht vorhanden. Weiterhin sind die Einträge nicht genügend unterschieden. Es sind z.B. keine ausreichenden Kriterien vorhanden, die zwischen Circuits und Dataflow einen Unterschied aufzeigen. Diese zwei Programmiermodelle können zurzeit nur anhand der ihnen zugeordneten Implementierungen differenziert werden.

Name	Class of interaction	Class of decomposition	Example implementations
...			
Circuits	Message Passing	Task	Verilog, VHDL
Dataflow	Message Passing	Task	Lustre, TensorFlow
...			
PRAM	Shared Memory	Data	Cilk, CUDA, OpenMP
PGAS	Shared Memory, Distributed Memory	Data, Task	UPC, X10

Tabelle 1.1.: Ein Auschnitt aus der Übersicht über die Programmiermodelle von Wikipedia [Wik16]. Diese Übersicht wurde durch einen Eintrag erweitert (rot).

Folgendes Zitat aus einem Survey über Programmiermodelle [KMZS08, Seite 2, Zeile 27ff] zeigt, dass auch Begriffe unterschiedlich genutzt werden:

„*Secondly, we present an investigation of* six parallel programming models *in the HPC community: three well established models (i.e.* Pthreads*,* OpenMP*, and* MPI*) and three relatively new models (i.e.* UPC*,* Fortress*, and* CUDA*).*“

In dem Zitat werden Pthrads, OopenMP, MPI, UPC, Fortress und CUDA als Programmiermodelle bezeichnet. Wie in der Tabelle 1.1 zu sehen, werden diese dort als Beispiele für Implementierungen aufgeführt. Programmiermodelle sind dagegen PRAM, Circuits oder Dataflow.

Die Tabelle 1.2 zeigt zusätzlich, dass auch komplett andere Strukturen (siehe Spalte Memory model) zu finden sind:

Language/ extension	Coordination abstraction	Type	Memory model	Deter- ministic	Embedding
MPI/PVM	low	task, data	msg pass (expl.)	no	library
...					
Java Threads	low	task, data	shared	no	library
...					
UPC	mid	task, data	PGAS	no	C extension
...					

Tabelle 1.2.: Ein Auschnitt aus der Übersicht über die Programmiermodelle aus einem Survey [BDT+13].

1.3. Ziel der Arbeit

Wie bereits erwähnt, existieren Versuche um die Programmiermodelle zu ordnen und zu unterscheiden. In dieser Arbeit soll darauf eingegangen werden. Herausgefunden werden soll, wie existierende Taxonomien aufgebaut sind. Stützend auf diesen gefundenen Versuchen der Darstellung soll eine einheitliche Taxonomie erstellt werden. Dabei wird versucht, die Übersicht der Programmiermodelle so umfassend wie möglich zu erstellen und ausreichende Unterscheidungskriterien zu nehmen. Die Implementierungen, die diese Programmiermodelle benutzen, werden auch verglichen. Aufbauend darauf wird versucht folgende Forschungsfrage zu beantworten:

- Wie können die Programmiermodelle der parallelen Programmierung einheitlich strukturiert werden und welche Unterscheidungsmerkmale besitzen sie?

1.4. Aufbau der Arbeit

Die Suche soll aufbauend auf dem Systematic Literature Review nach Kitchenham [KC07] durchgeführt werden (Kapitel 2). Ziel ist es, Taxonomien zu den Programmiermodellen zu finden und diese zu vergleichen. Das heißt, es werden die benutzten Begriffe verglichen und der Stand der Wissenschaft in diesem Bereich dargestellt. Daraufhin wird eine neue einheitliche Taxonomie erstellt, benutzte Begriffe werden einheitlich definiert, die Programmiermodelle und Implementierungen verglichen und Unterscheidungskriterien ausgearbeitet. Mit einer weiteren Kurzsuche soll herausgefunden werden, welche Hardwarekomponenten interessant für das jeweilige Programmiermodell sind. Ein Ausblick am Ende wird zeigen, ob eine ausreichende und komplette Taxonomie erstellt werden konnte.

2. Systematic Literature Review

In diesem Kapitel wird das Systematic Literatur Review nach Kitchenham [KC07] geplant und durchgeführt. Ziel ist es, alle relevanten Quellen über die Programmiermodelle der parallelen Programmierung zu erfassen und auszuwerten. Es werden dabei drei Phasen durchlaufen: *Planungsphase* (Abs. 2.1), *Durchführungsphase* (Abs. 2.2) und *Auswertungsphase* (Abs. 2.3).

2.1. Planungsphase

In der Planungsphase werden die Grundlagen der Suche ausgearbeitet. Weiterhin wird die Such-Strategie definiert, es werden Suchkriterien aufgestellt und Qualitätskriterien erarbeitet. Diese Phase stellt somit die Vorbereitung für die Ausführung der Suche in der nächsten Phase dar. Es werden dazu einige Kriterien, aufbauend auf dem Ziel der Suche, aufgestellt, nach denen einige Quellen aussortiert werden. Dies ist nötig, da nicht alle Ergebnisse der Suche garantiert relevant für die Arbeit sind.

2.1.1. Grundlagen

Bevor die SLR-Forschungsfragen ausgearbeitet und die Such- und Qualitätskriterien aufgestellt werden können, müssen der Hintergrund und die Grundlagen der Suche klar sein. Wie in der Einleitung (Abs. 1.1, S. 12) beschrieben, wird in dem Bereich der parallelen Programmierung gesucht. Es sollen die Quellen berücksichtigt werden, in denen es um die Programmiermodelle der parallelen Programmierung geht. Behalten werden insbesondere die Quellen, die einen Überblick bzw. ein Survey über oder eine Taxonomie für die Programmiermodelle beinhalten. Erwartet werden Programmiermodelle wie z.B. PGAS [PGA], PRAM [JaJ11] oder BSP [BSP]. Diese waren in vorher schon bekannten Quellen (z.B. [BDT$^+$13]) oft aufgeführt.

2.1.2. SLR Forschungsfragen

Abseits der schon formulierten Forschungsfrage dieser Arbeit in Abschnitt 1.3 werden noch einmal Forschungsfragen für das SLR formuliert. Sie stellen die Leitlinie des SLR dar und geben vor, nach welchen Gesichtspunkten die Quellen ausgewertet werden.

- Q_1 Welche Programmiermodelle für die parallele Programmierung werden aktuell (2006-2016) benutzt?
- Q_2 Auf welchen Konzepten bauen die zurzeit benutzten Implementierungen der Programmiermodelle auf?

Q_3 Welche Unterschiede gibt es hinsichtlich der verwendeten Begriffe und Unterscheidungskriterien in den Taxonomien der parallelen Programmiermodelle?

2.1.3. Protokoll

Such-Strategie: Es wird die Metasuchmaschine Google Scholar[1] benutzt. Kitchenham [KC07] führt diese mit in einer Liste auf, die für Software Engineering relevante Suchmaschinen enthält. Da Google Scholar eine Metasuchmaschine ist, wird sich in dieser Arbeit darauf beschränkt. Eine Testsuche bei ScienceDirect[2] und Keele University's electronic library[3] ergab z.B., dass dort keine weiteren relevanten Ergebnisse gefunden wurden.

Wie in der Motivation dieser Arbeit (Abs. 1.2) schon aufgezeigt wurde, werden Taxonomien und Übersichten benötigt. Aus diesem Grund wird natürlich auch die Suche danach ausgerichtet. Es wird erwartet, dass die SLR-Forschungsfragen (Abs. 2.1.2) besonders durch diese Art von Quellen beantwortet werden können. Populäre Programmiermodelle und Unterschiede der Klassifizierungen sowie Konzepten werden dort oft Bestandteil sein. Die Suchfragen werden in Englisch verfasst, da dies die Fachsprache in diesem Gebiet ist.

Die Suchfragen setzen sich folgendermaßen zusammen: Es werden Begriffe für Übersichten und Taxonomien mit Begriffen aus dem gesuchten Themenbereich verknüpft. Das heißt, Begriffe wie *Survey* oder *Review* verknüpft mit z.B. *Parallel Programming Model*, *Parallel Computing* oder auch *Parallel Model*. Diese Begriffe ergaben sich aus den Vorbereitungen für diese Arbeit. Dabei wurde nach allgemeinen Informationen über die Programmiermodelle gesucht. Durch die Vorbereitungen wurden auch schon erste Synonyme in der Bezeichnung entdeckt. Folgende Suchfragen ergaben sich aus den Vorüberlegungen:

S_1 *Survey AND „Parallel Programming Model“*

S_2 *Survey AND Models AND „Parallel Computation“*

Synonyme wurden zu den Begriffen *Survey* (*Review*, *Overview*), *Parallel Programming Model* (*Parallel Model*), *Parallel Computation* (*Parallel Computing*) und *Models* (*Model*) gefunden. Mit Hilfe dieser Synonyme konnten folgende Suchfragen aufgebaut werden. Dabei wurden die Fragen S_1 und S_2 weiterverwendet und die Synonyme dort eingesetzt:

$S_{1.2}$ *Survey AND „Parallel Programming Models“*

$S_{1.3}$ *Survey AND „Parallel Models“*

$S_{1.4}$ *Overview AND „Parallel Programming Models“*

[1] https://scholar.google.de/
[2] https://www.sciencedirect.com/
[3] opac.keele. ac.uk

$S_{2.2}$ *Review AND „Parallel Programming Models“*

$S_{2.3}$ *Survey AND Model AND „Parallel Computing“*

$S_{2.3}$ *Overview AND Model AND „Parallel Computation“*

Qualitätskriterien: Als nächster Schritt werden in einem SLR Qualitätskriterien aufgestellt. Um nur die Quellen zu behalten, die relevant für diese Arbeit sind, ist es notwendig alle gefundenen Quellen zu filtern. Sollten sie also mindestens eine der folgenden Ausschlusskriterien erfüllen, werden sie nicht in die Auswertung mit einbezogen. Wenn eine Quelle mindestens ein Einschlusskriterium und keins der Ausschlusskriterien erfüllt, wird es nicht ausgeschlossen.

Einschlusskriterien:

I_1 Quellen, die Programmiermodelle der parallelen Programmierung miteinander vergleichen.

I_2 Quellen, die mehrere Programmiermodelle der parallelen Programmierung vorstellen und beschreiben.

I_3 Quellen, die eine Taxonomie für die Programmiermodelle der parallen Programmierung vorstellen.

Ausschlusskriterien:

E_1 Quellen, die die Suchkriterien nicht erfüllen.

E_2 Quellen, die in anderen Sprachen als Deutsch und Englisch verfasst sind.

E_3 Präsentationsfolien (z.B. PowerPoint), da diese keine wissenschaftliche Arbeit darstellen.

E_4 Quellen, die über dem Zugang der TU-Chemnitz nicht erreichbar oder nicht verfügbar sind.

E_5 Quellen, die nicht auf die parallele Programmierung eingehen.

E_6 Quellen, die nicht mehrere Programmiermodelle vorstellen, da für die SLR-Forschungsfragen Taxonomien und Übersichten gesucht werden.

Informationsentnahme: Anschließend an die Anwendung der Kriterien, werden in dem SLR die übrigen Quellen analysiert. Dafür werden bestimmte Informationen in einer Tabelle gesammelt. Informationen sind unter anderem Autoren, Erscheinungsjahr oder Kurzbeschreibung des Inhalts. Am Ende werden die Quellen analysiert und die Ergebnisse niedergeschrieben.

2.2. Durchführungsphase

Nachdem in dem vorigen Abschnitt alle Vorbereitungen getroffen wurden, kann jetzt die Suche durchgeführt werden.

2.2.1. Ausführung der Suche

Aufbauend auf die Planungsphase (Abs. 2.1) wurde am 11.08.2016 die Suche ausgeführt. Das heißt, es wurden die Such-Fragen aus dem Abschnitt 2.1.3 in Google Scholar[4] eingegeben. Nach dem Eingeben und dem Durchsichten der Ergebnisse wurde herausgefunden, dass die gefundenen und relevanten Quellen schon im Titel darauf hinweisen, dass es sich um Surveys, Reviews oder Overviews handelt. Dieses Wissen konnte durch die erweiterte Suche bei Google Scholar auf die Suchfragen übertragen werden. Die Suchfragen wurden also noch einmal spezialisiert. Das heißt es wurden die Begriffe *allintitle:* und *intitle:* in Google Scholar benutzt. Dadurch wird der Titel nach den eingegebenen Begriffen durchsucht.

Weiterhin wurden die Ergebnisse auf den Zeitraum von 2006 - 2016 eingeschränkt, da in dieser Arbeit aktuelle Programmiermodelle betrachtet werden. Die Suche führte zu insgesamt 20 Ergebnissen (siehe Tabelle A.1 abzüglich FILTER 1 aus der nachfolgenden Aufzählung). Nach Anwendung der ersten Suchkriterien blieben 13 Ergebnisse übrig. Die Quelle [Li11] wurde vorerst ausgeschlossen, da sie kostenpflichtig ist. Nach Anfrage bei den Herausgebern von ASME Digital Collection [5] wurde dieses Paper jedoch freundlicherweise für diese Arbeit freigegeben. Diese restlichen Quellen wurden durch folgende Filter ausgeschlossen:

FILTER 1 Drei Quellen wurden ausgeschlossen, da die Quelle [Jek11] insgesamt viermal gefunden wurde. Zweimal als *Technischer Bericht* und zweimal als *Zweiter Forschungsbericht* [JB].

FILTER E_2 Die Quelle [JAXC+13] wurde ausgeschlossen, da sie in Mandarin verfasst ist.

FILTER E_3 Die Quelle [Owe08] wurde ausgeschlossen, da es sich um eine PowerPoint Präsentation handelte.

FILTER E_4 Die Quelle [LCW+11] wurde ausgeschlossen, da eine Anmeldung auf einer chinesischen Webseite erforderlich war.

FILTER E_5 Die Quelle [LT14] wurde ausgeschlossen, da sie nicht auf die Parallele Programmierung eingeht.

[4] http://scholar.google.de/
[5] https://www.asme.org/

2.2.2. Informationsentnahme

Nach dem Filtern, wurden die Quellen analysiert, um zu Beginn grundlegende Informationen zu erhalten. Die meisten Quellen gehören zu der Domäne High performance Computing (HPC). Alle Quellen wurden also überflogen und grundlegende Informationen extrahiert. Hier wurden weitere zwei Quellen ([LHL+15, DCH06]) ausgeschlossen, da ihr Inhalt mit dem Ausschlusskriterium E_6 (behandeln nur ein Programmiermodell) übereinstimmte. Die nachfolgende Tabelle 2.1 enthält die analysierten elf Quellen. Die Tabelle enthält allgemeine Informationen, wie den Titel, die Autoren, die Seitenzahl sowie die möglichen Klassifizierungen. Besonders relevant sind die Spalten *Modelle* und *Bemerkungen*. In der ersten wurden alle in der Quelle behandelten Programmiermodelle aufgezählt. Auch wenn der Begriff Programmiermodelle nicht konsequent mit dem gleichen Sinn benutzt wird. In der Spalte *Bemerkungen* finden sich Informationen über mögliche Klassifizierungen oder Besonderheiten der Quellen. In den Spalten finden sich somit schon erste Ansätze zur Beantwortung der SLR-Forschungsfragen Q_1 (aktuelle Programmiermodelle) und Q_3 (unterschiedlicher Übersichtsaufbau). Für Q_2 sowie einer ausführlichen Beantwortung der beiden anderen Fragen, müssen die Quellen im nächsten Schritt einzeln ausgewertet werden.

Titel	Autoren	Klass.	Seiten	Bemerkungen
A Survey of High-Level Parallel Programming Models [BDT+13]	Evgenij Belikov, Pantazis Deligiannis, Prabhat Totoo, Malak Aljabri, Hans-Wolfgang Loidl	ja	46	Klassifizierung nach Abstrahierung; für Einsteiger in die parallele Programmierung
Models for Parallel Computing: Review and Perspectives [KK07]	Christoph Kessler, Jörg Keller	nein	17	referenzieren ausführliche Surveys (1989-2000); beschreibt einzelne Modelle einheitlich
Models for Parallel Computing Review and Perspectives [Aga14]	Nahar hansmukh prayank, Bhavsar Dharmeshkumar Bhalchandra, Vishal Bhatnagar, Richa Tomer, S.K. Agarwal	nein	7	gekürzte Version des Papers von Christoph Kessler und Jörg Keller [KK07]
Models of parallel computation: a survey and classification [ZCSM07]	Zhang Yunquan, Chen Guoliang, Sun Guangzhong, Miao Qiankun	ja	10	Klassifizierung in drei Generationen; Vergleich BSP – LogP; viele Erweiterungen beschrieben

Tabelle 2.1 – Fortführung auf nächster Seite

Tabelle 2.1 – Fortführung von voriger Seite

Titel	Autoren	Klass.	Seiten	Bemerkungen
Survey of the sequential and parallel models of computation [Jek11]	Matevz Jekovec	ja	43	Klassifizierung nach: shared – distributed – hierarchical memory und Grad der Abstrahierung; alle Modelle ausführlich beschrieben; enthält charakteristische Eigenschaften von Modellen
Survey on Parallel Programming Model [KMZS08]	Henry Kasim, Verdi March, Rita Zhang, Simon See	nein	10	Vergleich von 6 Modellen anhand von 7 Kriterien
A Review of CUDA, MapReduce, and Pthreads Parallel Computing Models [MHCS14]	Kato Mivule, Benjamin Harvey, Crystal Cobb, Hoda El Sayed	nein	10	einfache Übersicht + Grundlagen GPU;
A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era [DMCN12]	Javier Diaz, Camelia Munoz-Caro, Alfonso Nino	ja	18	Klassifizierung: shared – distributed – heterogeneous parallel programming models - PGAS; enthält parallele Sprachen; Einblick in Hybride mit OpenMP, MPI, Pthreads, CUDA
A Review on New Paradigm's of Parallel Programming Models in High Performance Computing [RGP12]	Amitkumar S. Manekar, Pankaj Kawadkar, Malati Nagle	ja	4	Fehler in Grammatik und Sprache; dem Paper von Javier Diaz et al. [DMCN12] fast gleich
Recent Advancements in Parallel Algorithms for String Matching on Computing Models – A Survey and Experimental Results [SRA+11]	Chinta Someswararao, Butchi Raju, S. Appaji, Viswanadha Raju, K. Reddy	ja	9	Klassifizierung: 1) Intrusion Detection Systems 2) PRAM Model 3) Other; Parallele Algorithmen zum String Matching Problem; wenig Relevanz
A Survey of Task-Based Parallel Programming Models [Li11]	Xin Li	ja	4	Klassifizierung: 1) Control-Driven Models 2) Data-Driven Models; ausführliche Beschreibung von 7 Modellen

Tabelle 2.1.: Alle ausgewerteten Quellen aufgelistet, mit zusätzlichen Informationen, wie eine möglicherweise vorhandene Klassifizierung der Programmiermodelle und interessante Details in den Bemerkungen.

In der ersten Informationsentnahme zeigte sich, dass in zwei Fällen sich jeweils zwei Quellen in der Struktur und Inhalt sehr ähneln, obwohl jeweils andere Autoren genannt sind. Der Unterschied liegt jeweils darin, dass eine Quelle einen viel größeren Umfang aufweist und somit die gleichen Abschnitte mit mehr Tiefe behandelt. Die andere Quelle kann damit nur noch als Kurzfassung dienen, ohne neue Inhalte zu präsentieren.

2.3. Auswertungsphase

In dieser Phase werden die Quellen einzeln ausgewertet, indem die SLR-Forschungsfragen (siehe 2.1.2) die Leitlinien bilden. Das heißt, wichtige Punkte bei der Auswertung sind die vorgestellten Programmiermodelle sowie die vorgenommene Klassifizierung, falls eine vorhanden ist.

2.3.1. Auswertung der Quellen

Behandelte Programmiermodelle: In der Tabelle 2.2 sind alle Programmiermodelle zu finden, die in den einzelnen Quellen vorkommen. Dabei wurden immer die Modelle ausgewählt, die in den Quellen als Programmiermodelle bezeichnet sind. Einige sind jedoch keine Programmiermodelle nach der Definition dieses Begriffs in dieser Arbeit. Im Folgenden werden diese zur Unterscheidung auch als Sprachen bezeichnet. Zu sehen ist hier schon, dass einige Programmiermodelle und Sprachen mehrfach auftauchen. Damit kann aus dieser Tabelle schon eine Antwort auf die SLR-Forschungsfrage Q_1 abgeleitet werden: Am häufigsten finden sich die Programmiermodelle MPI/MessagePassing (6x), PRAM(5x), OpenMP(5x), CUDA(4x) und BSP(4x).

Die erste Zeile zeigt deutlich, wie in manchen Quellen Modelle, Sprachen und Bibliotheken unter dem Begriff Programmiermodelle zusammengefasst wurden. Einige Quellen haben die aufgeführten Programmiermodelle und Sprachen auch klassifiziert. Genauere Details werden jedoch erst im nächsten Schritt sichtbar. In diesem werden die Quellen einzeln und ausführlich nach den SLR-Forschungsfragen ausgewertet.

Programmiermodelle und Sprachen	Quellen											
	[BDT+13]	[KK07]	[Aga14]	[ZCSM07]	[Jek11]	[KMZS08]	[MHCS14]	[DMCN12]	[RGP12]	[SRA+11]	[Li11]	Summe
MPI	✓	✓	✓			✓		✓	✓			6
PRAM		✓	✓	✓	✓					✓		5
OpenMP	✓					✓		✓	✓		✓	5
BSP		✓	✓	✓	✓							4
Pthreads	✓					✓	✓	✓				4
CUDA	✓					✓	✓	✓				4
PGAS		✓	✓					✓				3
OpenCL	✓							✓			✓	3
ArBB	✓							✓				2
TBB	✓										✓	2
MapReduce	✓						✓					2
Fortress	✓					✓						2
UPC	✓					✓						2
Cilk	✓										✓	2
Direct Compute								✓				1
Jade											✓	1
SMPSs											✓	1
OoOJava											✓	1
LogP				✓								1
CLUMPS				✓								1
NHBL				✓								1
P-HMM&BT				✓								1
UMH				✓								1
DRAM				✓								1
HPM				✓								1
PMH					✓							1
PCO					✓							1

Tabelle 2.2.: Übersicht zur Veranschaulichung des häufigen Auftretens einiger Modelle und Sprachen. Belikov et al. [BDT+13] erwähnt noch TPL, SAC, HPF, DPH, PLINQ, CAF, Chapel, X10, CnC, Parallel Haskell, Erlang, Manticore, Renderscript, C++AMP, Offload, SkePU und P3L. Bei Zhang Yunquan et al. [ZCSM07] tauchen noch QSM, DMM, Postal, Atomic, C^3 und BDM auf.

Quellen mit einer Klassifizierung der Programmiermodelle

In sieben von elf Quellen sind die aufgeführten Programmiermodelle in Klassen geordnet. Diese Ordnungen sind wichtig für die SLR-Forschungsfrage Q_3, um herauszufinden, welche Begriffe und Unterscheidungskriterien für eine Taxonomie verwendet werden. Im Folgenden sind die einzelnen Paper aufgeführt:

In dem Paper von Belikov et al. [BDT+13] werden zwei Klassifizierungen vorgestellt. Es wird bemerkt, dass es schwierig sei eine Klassifizierung zu erstellen, da es viele verschiedene aber auch sinnvolle Wege gäbe. In dem Diagramm des Papers wird die erste Klassifizierung gezeigt (Tabelle 2.1).

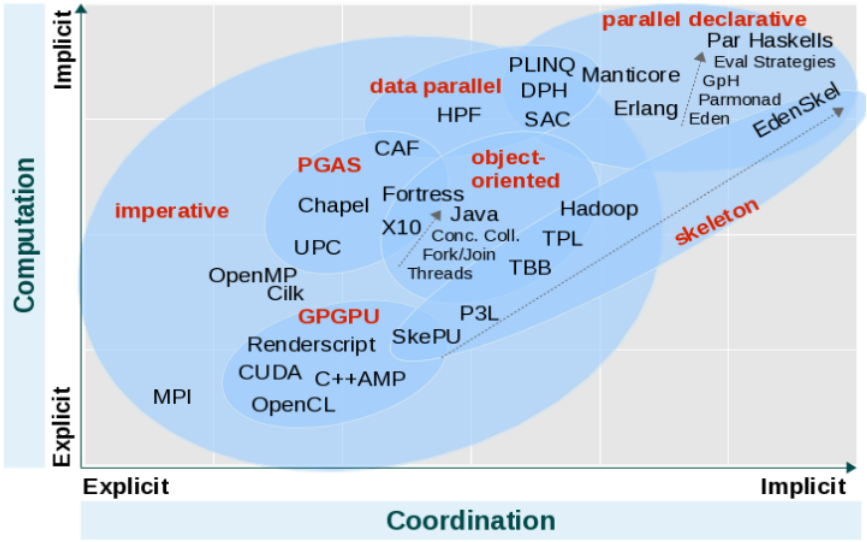

Abbildung 2.1.: Diagramm über die Programmiermodelle aus dem Paper von Belikov et al. [BDT+13].

Die insgesamt 26 Programmiermodelle in dem Paper werden in zwei Dimensionen gezeichnet. Diese sind *Berechnung* (Computation) und *Koordination* (Coordination) und reichen jeweils von explizit zu implizit. *Berechnung* bezieht sich z.B. auf die algorithmischen Lösungen und *Koordination* beschreibt z.B. die Art und Weise, wie Prozesse kommunizieren und Zugriff auf geteilte Ressourcen bekommen. Nach diesem Diagramm findet sich z.B. Par Haskells am äußersten Platz, da die Berechnung und Koordination beide sehr implizit eingeschätzt wurden. MPI befindet sich im Gegensatz dazu nahe der Schneidung der Koordinaten, da dort beides explizit gehandhabt wird. Die zweite Klassifizierung wird auch schon in dem Diagramm angedeutet. Für diese Klassifizierung werden die Programmiermodelle in sieben Gruppen aufgeteilt. Die genauere Aufteilung wird in einer Tabelle visualisiert. Die sieben Klassen sind: *Parallel Imperative*, *Parallel Object-Oriented*, *Data Parallel*, *PGAS*, *Parallel Declarative*, *GPU Programming*, *Algorithmic Skeletons*. Das Paper bietet durch Kapitel über momentane Trends in der Parallelisierung, durch eine Einführung in Herausfor-

derungen und durch abschließende Fallstudien einen breiten Einblick in die parallele Programmierung. Die Autoren folgern am Ende, dass die Programmiermodelle vermehrt implizite Berechnung und Koordination aufweisen und dass rein deklarative Programmiermodelle am besten zu parallelen Berechnungen passen würden.

Zhang Yunquan et al. ordnet in sinem Paper [ZCSM07] die behandelten Programmiermodelle in drei Generationen. Diese Generationen sind chronologisch nach ihrem ersten Auftreten sortiert. Die erste umfasst *shared memory parallel computational models* (1978), die zweite *distributed memory parallel computational models* (1984) und die dritte *hierarchical memory parallel computational models* (1993). In den Generationen werden durchschnittlich fünf Programmiermodelle ausführlich beschrieben. Es wird auf Synchronisation, Kommunikation und Speicherhierarchie eingegangen. Auch viele Erweiterungen der Programmiermodelle werden behandelt. Die drei Generationen werden anschließend noch miteinander verglichen. Die Autoren beobachten, dass mit der Zeit immer mehr Parameter (z.B. L in LogP als Parameter für die Latenz) in die Modelle integriert wurden. Weiterhin wären in späteren Programmiermodellen verschiedene vorige Programmiermodelle kombiniert. Die Autoren können auch beobachten, dass hierarchische Speichermodelle global (Fokus auf hierarchische Parallelisierung) oder lokal (Fokus auf lokale Speicherkosten) auftreten. Es wird empfohlen Wege zu finden, um diese zwei in der Zukunft zu vereinen. Weiterhin weisen die Autoren darauf hin, dass durch die immer steigende Komplexität es schwerer wird diese Modelle zu analysieren. Auf die Entwicklung von automatischen Werkzeugen zum Extrahieren der Parameter aus Algorithmen wird deshalb hingewiesen.

Matevz Jekovec [Jek11] differenziert am Anfang seines Papers den Begriff der Modelle in drei Bereiche. 1. *model of computation*, 2. *programming model*, 3. *hardware model*. Er ordnet anschließend vier *models of computation* mit deren Erweiterungen sowie ihren sequentiellen Gegenstücken in einen Strukturbaum. Die Programmiermodelle sind nach ihrem Abstraktionsgrad (große Abstrahierung bis sehr realistisch) geordnet. Die Modelle sind auch nach ihrem Alter sortiert. Diese Sortierung entspricht der Klassifizierung im vorigen beschriebenen Paper [ZCSM07]. Das heißt es existieren drei Generationen: *shared-*, *distributed-* und *hierarchical memory parallel computations models*. Der Autor baut das Paper nach der gleichen Ordnung wie die des Strukturbaumes auf. Als erstes die sequentiellen Programmiermodelle, dann ihre Gegenstücke im parallelen Bereich. Der Autor beschränkt sich dabei auf die wichtigsten Programmiermodelle im Laufe der Zeit. Anschließend wird das Programmiermodell Vector RAM beschrieben. VRAM verbindet sequentielle und parallele Programmiermodelle. In der Zusammenfassung bewertet der Autor einzelne beschriebene Programmiermodelle. Allgemein zeigt der Autor einen tiefen Einblick in die Programmiermodelle. Dadurch können Unterscheidungsmerkmale zwischen den Modellen für die SLR-Forschungsfrage Q_3 identifiziert werden. Er benutzt Messmethoden und geht nach verschiedenen Charakteristiken von parallelen Programmiermodellen. Z.B. sind die Umgebungscharakteristiken für parallele Programmiermodelle nach Maggs, Matheson und Tarjan [MMT95]: *Memory access*, *Sysnchronization*, *Latency*, *Bandwidth*

und *Primitives*.

In einem weiteren Paper, von den Autoren Javier Diaz et al. [DMCN12], werden dominante (populäre) Programmiermodelle betrachtet. Das wären OpenMP für geteilten und MPI für gemeinsamen Speicher. Die Autoren definieren dadurch *Pure Parallel Programming Models*. Im Laufe der Zeit wären aber heterogene Programmiermodelle entstanden, die sich davon unterscheiden. Weiterhin führten Simulationen von globalem Speicherplatz in einem Umfeld mit verteiltem Speicher zu PGAS. Am Ende des Papers werden noch Hybride Programmiermodelle vorgestellt, die sich aus den vorigen genannten Modellen zusammensetzen. Das führt somit zu einer Klassifizierung der Programmiermodelle in fünf Gruppen: *Shared Memory*, *Distributed Memory*, *Heterogeneous Parallel Programming Models*, *PGAS* und *Hybrid (distributed|shared Memory + GPU) models*. Heterogene parallele Programmiermodelle meint z.B. OpenCL und CUDA. In dem Paper beschreiben die Autoren die einzelnen Programmiermodelle ausführlich und beschäftigen sich anschließend mit *Distributed Programming*. Am Ende werden Sprachen vorgestellt, die für die parallele Programmierung geeignet sind. Die Autoren beobachten, das OpenMP, CUDA, MPI und OpenCL dominierend sind. MPI sticht dabei heraus. Da der Ansatz des geteilten Speichers der populärste wäre, ist MPI der Standard in der HPC-Community geworden. Diese Beobachtungen reichen von 2000-2010. Weiterhin folgern die Autoren, dass für heterogene Modelle OpenCL dominant werde und allgemein der Open Standard Ansatz wahrscheinlich das Ziel sein wird. Dadurch könnten die Modelle verbreitet werden und eine Einheitlichkeit geschaffen werden. Der Inhalt des Papers von Amitkumar S Manekar et al. [RGP12] weist einen sehr ähnlichen Inhalt auf und somit auch die selbe Klassifizierung.

In dem Paper von Chinta Someswararao [SRA+11] wird die Klassifizierung in 1. *Intrusion Detection System Models*, *PRAM Models* und *Other Approaches* unterteilt. Das Paper beschäftigt sich aber besonders mit parallelen Algorithmen für das String Matching-Problem, die hier nicht interessant sind.

Xin Li beschränkt sich in dem Paper [Li11] auf task-based Programming Models. Die insgesamt 7 Programmiermodelle werden in zwei Kategorien geordnet: Die erste Kategorie umfasst die Control-Driven Modelle. In der zweiten Kategorie finden sich die Data-Driven Modelle. Die insgesamt 7 Programmiermodelle sind hauptsächlich Bibliotheken oder Programmiersprachen. Diese Modelle werden in dem Paper ausführlich beschrieben. Am Ende gibt es einen Ausblick mit verschiedenen Vorschlägen zur Verbesserung und Weiterführung der Task-Based Programming Models.

Quellen ohne Klassifizierung

Die restlichen vier Quellen klassifizieren die Programmiermodelle nicht, sondern stellen sie unsortiert einzeln vor. Das heißt, wichtige Punkte sind hier besonders die Art und Weise, wie sie Programmiermodelle unterscheiden und welche Kriterien zur

Bewertung genommen werden. Die SLR-Forschungsfragen Q_2 und ein Teil von Q_3 könnten damit beantwortet werden.

Christoph Kessler und Jörg Keller präsentieren mit ihrem Paper [KK07] ein sehr umfassendes Review über populäre und zukunfts-relevante Programmiermodelle. Sie beschreiben die Modelle zuerst allgemein, dann in Bezug auf ihre praktische Relevanz und zuletzt stellen sie die dazugehörigen Implementierungen vor. Zu der Beschreibung eines *Models of Computation* gehört zum einen das *Parallel Programming Model* und zum anderen das *Cost Model* (berechnet Laufzeit und Ressourcennutzung). Zu den Programmiermodellen gehört auch meistens ein *Memory Model*. Am Ende des Papers stellen die Autoren noch Methodiken zur parallelen Programmierung vor und fassen dann zusammen, dass Computerarchitekturen vermehrt hybrid entworfen werden. Das Paper von Nahar hansmukh prayank et al. [Aga14] stellt eine verkürzte Version dieses Papers dar und liefert dadurch keine neuen Inhalte.

Die Autoren Henry Kasim et al. stellen in ihrem Paper [KMZS08] eine Methodik dar, mit der parallele Programmiermodelle evaluiert werden können. Dazu stellen sie sieben Kriterien vor und wenden diese an sechs Programmiermodelle an. Die Kriterien sind: 1. *system architecture* (gemeinsamer oder geteilter Speicher); 2. *programming methodologies* (API, neue Spezifikationen); 3. *worker management* (wie Erstellung von worker/threads/processors?); 4. *workload partitioning scheme* (wie ist die Arbeit aufgeteilt?); 5. *task-to-worker mapping* (wie sind tasks zu worker gemappt?); 6. *synchronization* (wann haben worker Zugriff auf geteiltem Speicher?); 7. *communication model* (welches wird genutzt?). Das Ziel der Autoren ist es mit diesen Kriterien eine grundlegende Richtlinie zu schaffen, mit der Programmiermodelle evaluiert werden können. Die Autoren fassen jedoch auch zusammen, dass diese Kriterien nicht umfassend sind. Um eine ausführende Qualitätsbestimmung zu erstellen, müsse z.B. auch der Debugging Support des Programmiermodelles beachtet werden.

In dem letzen Paper von Kato Mivule et al. [MHCS14] findet sich ein ausführlicher Überblick über die Programmiermodelle CUDA, MapReduce und Pthreads. Die Autoren haben im Vorfeld dazu noch eine Einführung in die parallele GPU Programmierung eingefügt. Die Funktionsweise der Programmiermodelle werden umfassend beschrieben, so dass die dahinter liegenden Konzepte klar werden. Dem Lesenden soll dadurch ein schnellerer Einblick in parallele Programmierkonzepte gegeben werden, so dass dieser bessere Implementierungen in großen Projekten erstellen kann. Dies stellt gemäß den Autoren das Ziel dieses Papers dar.

2.3.2. Zusammenfassung der Ergebnisse

Am Ende des SLR können nun auch die SLR-Forschungsfragen beantwortet und die Suche bewertet werden. Insgesamt wurden 11 Quellen ausgewertet. In 7 von diesen Quellen wurden Klassifizierungen gefunden, die in den nachfolgenden Kapiteln eine große Rolle spielen werden.

Die SLR-Forschungsfrage Q_1 wurde schon mit der Tabelle 2.2 beantwortet. Sie lautete: *Welche Programmiermodelle für die parallele Programmierung werden aktuell (2006-2016) genutzt?* In der Tabelle sind alle gefundenen Modelle aufgezählt und es kann abgelesen werden, welche davon am häufigsten erwähnt werden. MPI wird in 6 von 11 Quellen vorgestellt, gefolgt von PRAM mit 5 Vorkommnissen und BSP, CUDA und OpenMP mit jeweils 4 Vorkommnissen. Die Quellen sind aus der Zeit von 2007-2014. Eine Liste mit allen vorkommenden Programmiermodellen mit kurzer erklärender Beschreibung findet sich im Anhang. Alle dort aufgelisteten Programmiermodelle wurden in den verschiedenen Quellen als Programmiermodelle vorgestellt. Die Liste ist geordnet nach Programmiersprachen, Bibliotheken und Programmiermodellen (nach der Definition dieser Arbeit).

Die zweite SLR-Forschungsfrage richtet sich an die Konzepte der Implementierungen. Die Frage lautet: *Auf welchen Konzepten bauen die zurzeit benutzten Implementierungen der Programmiermodelle auf?* Javier Diaz et al. schreibt [DMCN12], dass die meisten Programmiersprachen (mit Möglichkeiten zu parallelen Programmierung) sich an HPC orientieren. Sie können in Shared- und Distributed Memory Models unterteilt werden. OpenMP basiert z.B. auf dem Shared Memory Model und das PRAM Model kann in OpenMP implementiert werden. Die Programmiersprache Cilk ist eine imperative Sprache und basiert, genau wie OpenMP, auf dem Shared Memory Model [DMCN12]. Cilk baut auf C auf. Im Anhang sind die wichtigsten gefundenen Modelle, Sprachen und Bibliotheken aufgelistet. Dort sind auch die Informationen über zugrunde liegende Modelle und Ansätze (siehe Anhangs-Kapitel B, S.50).

Um die dritte Forschungsfrage zu beantworten, müssen vor allem die Klassifizierungen in den Quellen angeschaut werden. Die dritte Frage lautete: *Welche Unterschiede gibt es hinsichtlich der verwendeten Begriffe und Unterscheidungskriterien in den Taxonomien der parallelen Programmiermodelle?* Zu den Begriffen: Parallele Programmiersprachen und Bibliotheken werden oft unter dem Sammelbegriff parallele Programmiermodelle vereint. Dies sind vermehrt OpenMP, MPI, OpenCL, usw. PGAS wird in manchen Quellen auch unter die Speichermodelle gezählt. Dort sind die Modelle in Programmiermodelle, Kostenmodelle und Speichermodelle unterteilt. Eine gegliederte Auflistung der Begriffe für jede gefundene Taxonomie ist im Kapitel 3 zu finden. Es finden sich also oft Unterschiede in der Benutzung des Begriffs Programmiermodelle. Zu den Unterscheidungskriterien: Die Differenzierung der Modelle findet auch auf verschiedene Art und Weise statt. Einige Quellen unterscheiden nach dem grundlegenden Speicherkonzept (Distributed oder Shared Memory, andere unterscheiden zwischen Control-Driven- und Data-Driven Models, andere zwischen GPGPU, klassischen und Hyriden und wieder andere zwischen einer expliziten oder impliziten Abstraktion zum Programmierer. Hier wurden wenige Gemeinsamkeiten gefunden.

3. Existierende Taxonomien

In diesem Kapitel werden die gefundenen Taxonomien aus den ausgewerteten Papern beschrieben und dargestellt. Dann kann ausgemacht werden, wo Gemeinsamkeiten und Unterschiede liegen und was der Grund dafür ist.

3.1. Xin Li

In dem Paper [Li11] werden nur Task-Based Parallel Programming Models betrachtet. Diese werden in zwei Kategorien unterteilt.

Begriffe

- **Parallel Programming Models** meint Programmierschnittstellen, Bibliotheken und Programmiermodelle.
- **Parallel Programming Languages** meint Programmiersprachen, wie z.B. Jade.
- **Task-Based Models** sind Modelle, bei denen Tasks parallel ablaufen, im Gegensatz zu Data Parallel Models. Dort existieren Data-Sets auf denen Tasks gemeinsam zugreifen.
- **Control-Driven** sagt aus, dass die Ablaufreihenfolge in einem Programm auch in dem Programm festgelegt ist.
- **Data-Driven** sagt aus, dass Tasks ausgeführt werden, wenn alle Eingabedaten vorhanden sind. Es ist somit Daten-abhängig.

Klassifizierung

Das Paper beschränkt sich auf Task-Based Parallel Programming Models. Damit nimmt es schon eine Unterteilung der Programmiermodelle vor. In Task-Based Models und Data Parallel Models. Weiterhin werden die Task-Based Models in zwei weitere Kategorien unterteilt. Control-Driven Models bezeichnet Modelle, wie OpenMP. Hier muss der Programmierer z.B. sicherstellen, dass keine Datenabhängigkeiten zwischen den parallel ausgeführten Tasks existieren. Jade ist im Gegensatz dazu ein Data-Driven Modell. Bei Jade muss der Programmierer die Datenzugriffs-Methode für jeden Task erstellen, da Jade diese Informationen benötigt.

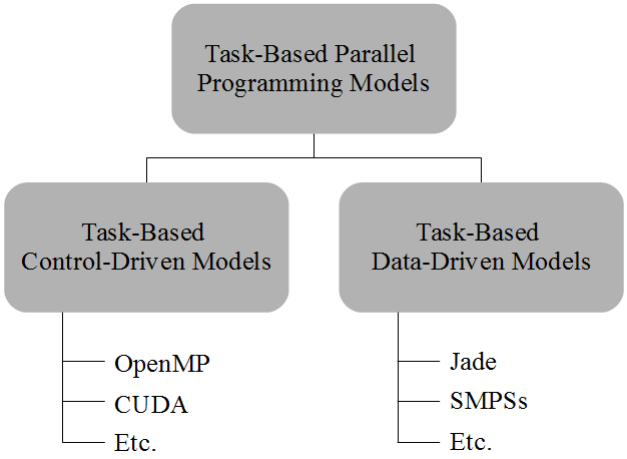

Abbildung 3.1.: Unterteilung der Task-Based Models in zwei Kategorien. Zu jeder Kategorie gehören mehrere Sprachen, Bibliotheken oder Programmiermodelle.

Bemerkungen

Die Programmiersprachen, Modelle, etc. werden alle unter einem Begriff vereint. Die Klassifizierung beschränkt sich natürlich nur auf den Bereich der Task-Based Parallel Programming Models, die Kategorien control-/ data-driven sind jedoch nicht darauf beschränkt.

3.2. Zhang Yunquan et al.

In dem Paper [ZCSM07] werden drei Generationen von Programmiermodellen untersucht.

Begriffe

- **Parallel Computational Models** meint hier die Modelle, die auch nach der Definition in dieser Arbeit als Programmiermodelle bezeichnet werden.

Klassifizierung

Die drei Unterteilungen der Programmiermodelle sind zeitlich geordnet und gegründet. Die drei Generationen haben also einen zeitlichen Startpunkt. Als erstes kommen die Modelle mit Speichermodellen, die auf gemeinsamen Speicher aufbauen. Die Modelle aus der zweiten Generation bauen auf verteiltem Speicher auf. In der dritten Generation ist der Speicher in Hierarchien aufgebaut. Diese haben unterschiedliche Zugriffszeiten. In der dritten Generation finden sich vermehrt Erweiterungen von Programmiermodellen aus den ersten beiden Generationen.

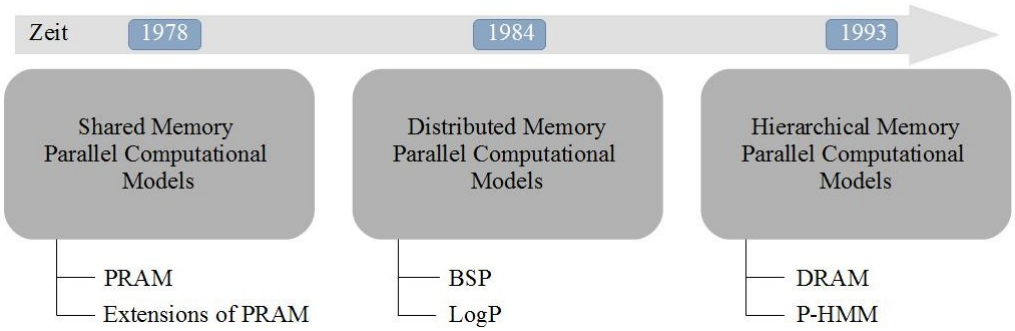

Abbildung 3.2.: Chronologische Klassifizierung der Modelle fokussierend auf den jeweiligen Speichermodellen.

Bemerkungen

Das Paper erwähnt auch, dass es in der Zukunft wahrscheinlich unerlässlich wird, Modelle mit hierarchischen Speicher zu entwickeln. Damit wird die dritte Generation im Fokus sein. Eine Aufteilung der Programmiermodelle wäre mit dieser Klassifizierung somit in der Zukunft nicht mehr nützlich. Zu dem Zeitpunkt der Veröffentlichung des Papers (2012) finden sich noch die meisten, von den in dem Paper vorgestellten Programmiermodelle in der zweiten Generation.

3.3. Javier Diaz et al.

In dem Paper von Javier Diaz et al. [DMCN12] gehen die Autoren auf vier verschiedene Arten von parallelen Programmiermodellen ein.

Begriffe

- **Parallel Programming Models** beschreibt Bibliotheken und Programmiermodelle.
- **Classical Models** auch Pure Models genannt, bauen nur auf der klassischen Grundlage (shared- oder distributed memory) auf.
- **Heterogeneous Models** bauen auf dem GPGPU Konzept auf. Sie verbinden also die CPU mit der GPU.
- **Languages with Parallel Support** bezeichnen Sprachen, die parallele Programmiermodelle implementieren können.

Klassifizierung

Die Unterteilung findet in dem Paper hauptsächlich zwischen Shared-/Distributed Memory Models und Heterogeneous Models statt. PGAS und Hybrid Parallel Pro-

gramming Models bilden jeweils eine eigene Kategorie, da sie die ersten beiden Kategorien verbinden.

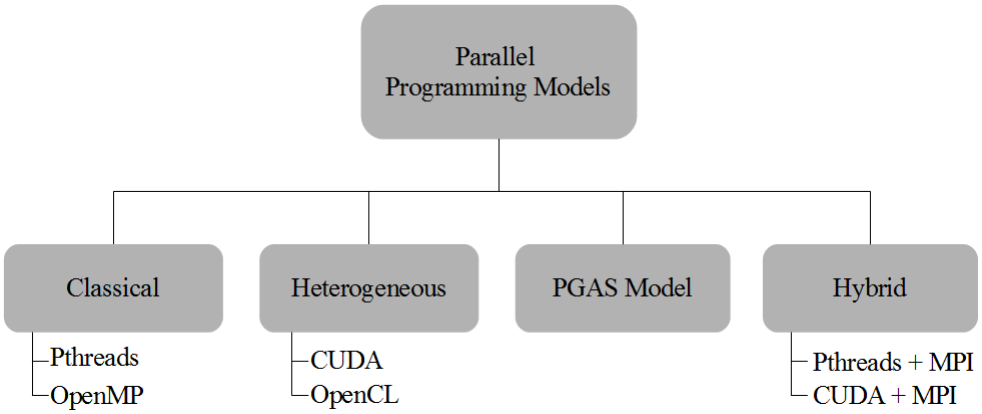

Abbildung 3.3.: Klassifizierung in vier Gruppen mit je ein paar Beispielen. In der Gruppe Hybrid wurden in dem Paper die in den ersten beiden Gruppen behandelten Modelle kombiniert.

Bemerkungen

Interessant ist hier die Aufteilung der Programmiermodelle. Es wird zwischen den Shared- oder Distributed Memory Ansätzen unterschieden. Heterogeneous Models, PGAS und Hybrid Programming Models, die andere Elemente enthalten oder Shared- und Distributed Memory verbinden, werden in dem Paper außerhalb betrachtet. Das Paper unterscheidet außerdem, im Gegensatz zu den anderen, zwischen Bibliotheken wie OpenMP und parallelen Sprachen. Die ersteren gehören unter dem Begriff Parallel Programming Models, während die zweiten unter Languages with Parallel Support aufgeführt werden.

3.4. Matevz Jekovec

In dem Paper [Jek11] stellt der Autor eine ausführliche Klassifizierung über viele Programmiermodelle vor. In dem Paper werden sowohl sequentielle als auch parallele Programmiermodelle behandelt.

Begriffe

- **Model of Computation** bezeichnet eine Engine zur Lösungsfindung. Auch als Abstract Machine, Cost Model oder Performance Model bezeichnet, da es auch technische Merkmale von Plattformen für die Berechnungen erfasst.
- **Parallel Programming Models** ist ein Modell, dass die Semantik für die Abstraktion zum Programmierer zur Verfügung stellt.

- **Hardware Model** ist eine Abstraktion, die für spezielle Sprachimplementationen genutzt wird.

Klassifizierung

Folgende Abbildung 3.4 stellt die eine Hälfte des Baumes aus dem Paper dar. Zusätzlich davon gibt es dort parallel die sequentiellen Modelle und dazwischen Vektor RAM als Hybrid.

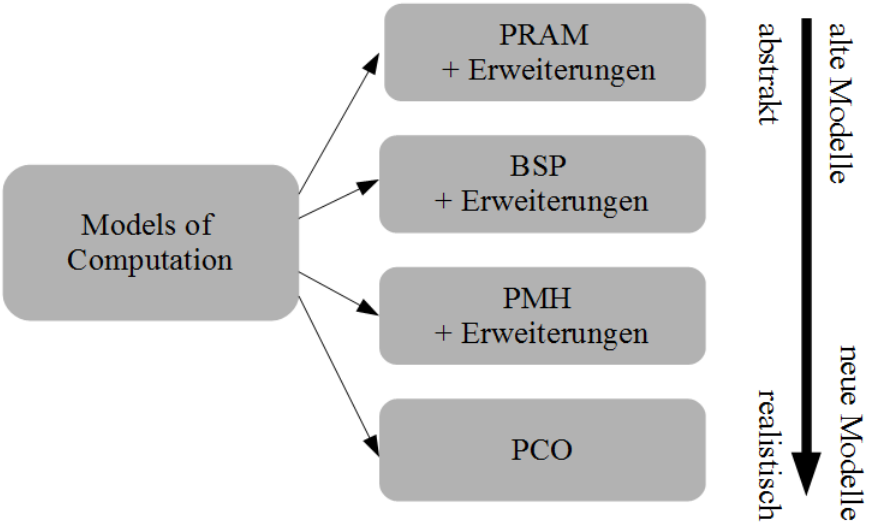

Abbildung 3.4.: Klassifizierung der Modelle von oben nach unten. Die abstraktesten und ältesten Modelle oben. Die realistischsten und neuesten unten.

Bemerkungen

Dieses Paper stellt gute Definitionen der genutzten Begriffe auf und benutzt diese nachvollziehbar und gut differenziert. Die Klassifizierung kombiniert außerdem eine chronologische Ordnung der Modelle mit der Ordnung nach abstrakten und realistischen Modellen. Die Modelle ordnen sich somit, wie in anderen Papern, in drei Generationen: Shared-, Distributed- und Hierarchical Parallel Programming Models. Der Autor zeigt außerdem, dass die behandelten Programmiermodelle immer realistischer werden. Das realistischste und neueste Programmiermodell ist somit das Parallel Cache-Oblivious Model (PCO). Das Paper stammt von 2011.

3.5. Evgenij Belikov et al.

Evgenij Belikov et al. stellen in dem Paper [BDT+13] über 20 Programmiermodelle vor und klassifizieren sie.

Begriffe

- **Parallel Programming Models** meint parallele Programmiersprachen, Erweiterungen und Bibliotheken.

Klassifizierung

Es gibt eine Ordnung der Programmiermodelle in Gruppen. In dem Paper wird diese Klassifizierung in einer Tabelle veranschaulicht.

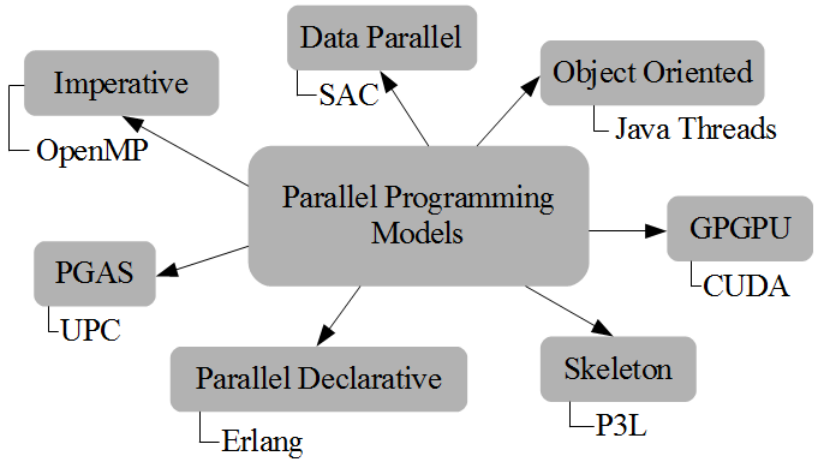

Abbildung 3.5.: Unterteilung der parallelen Programmiermodelle in insgesamt 7 Gruppen. Jeweils ein Beispiel ist angegeben.

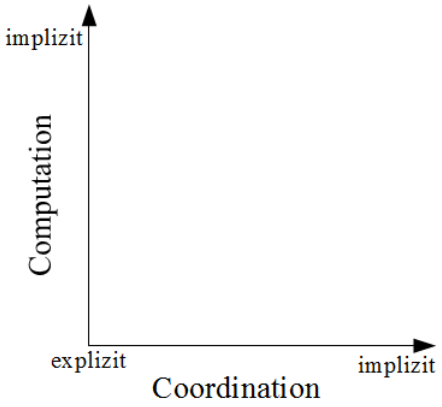

Abbildung 3.6.: Diagramm über zwei Dimensionen. Die Programmiermodelle werden nach ihrer expliziten/impliziten Behandlung von Coordination und Computation hier eingeordnet.

Zusätzlich dazu wurden die Gruppen als Cluster in einem Diagramm erweitert. In der Abbildung 2.1 (S. 24) ist die Klassifizierung zu sehen. Die Programmiermodelle

aus den Gruppen wurden nun von den Autoren in zwei Dimensionen eingeordnet. Diese Dimensionen sind Coordination und Computation. Sie reichen von explizit bis implizit. Die Achse Coordination beschreibt, in welchem Grad die Parallelisierung explizit oder implizit verwaltet wird (siehe Abbildung 3.6). Das heißt, ob der Programmierer z.B. Synchronisation oder Kommunikation selbst verwaltet (OpenMP) oder ob dies bis auf wenige Details abseits vom Programmierer geschieht (GpH). Ähnlich ist es bei der Achse Computation, bei der es um die algorithmischen Lösungswege geht.

Bemerkungen

Die vorgestellte Klassifizierung ist sehr ausführlich und ermöglicht eine gute Unterscheidung von vielen verschiedenen Programmiermodellen. Jedoch kommen in diesem Paper fast ausschließlich Programmiersprachen oder Erweiterungen vor, sodass herausgefunden muss, inwiefern sich diese Klassifizierung als ausreichend erweist. Die angegebenen Gruppen sind Programmierkonzepte: imperative-, objektorientierte-, data-parallel- sowie deklarative Sprachen. Dazu PGAS, Algorithmic Skeletons und GPU Programmierung.

3.6. Vergleich der existierenden Taxonomien

Die Taxonomien weisen einige Gemeinsamkeiten, aber auch viele Unterschiede auf. Hintergründe für die Unterschiede sind die verschiedenen Ziele, die mit den jeweiligen Papern erreicht werden sollten. Die eingeführten Taxonomien in den Papern sollten natürlich unterstützend auf dem Weg zum Ziel für die Autoren und den Lesern wirken.

In dem Paper von Xin Li [Li11] sind nur Task-Based Programming Models vorhanden, da der Ausblick auf automatische Analysen in APIs für task-level Compiler lag. Die weitere Aufteilung der Programmiermodelle Task-Based Programming Models in Control- und Data-Driven wurde durchgeführt, da der Fokus des Papers auf der Effizienz von Modellen liegt. Es wurde unter anderem geforscht welche Vorteile von unterschiedlichen Ausführungs-Mechanismen (Execution) in den Modellen kommen können. Die Aufteilung in Data-/Control-Driven macht somit Sinn, da dies eine Unterteilung der Modelle in verschiedene Ausführungs-Mechanismen ist.

Weiterhin wurde eine Aufteilung der Programmiermodelle in drei Generationen aufbauend auf den jeweilig genutzten Shared-, Disributed oder Hierarchical Memory Ansatzes gefunden [ZCSM07]. Das Ziel des Papers ist die Geschichte der Programmiermodelle vorzustellen. Dabei wurde der Fokus auf die Speichermodelle gelegt, da herausgefunden wurde, dass diese zeitlich aufeinanderfolgend entstanden. Ein weiterer Grund ist auch, dass so Programmiermodelle mit Hierarchical Memory Model Ansatz vorgestellt wurden, welche laut den Autoren jetzt und vermehrt in Zukunft eine tragende Rolle beinhalten.

Gemeinsamkeiten zur Klassifzierung von Zhang Yunguan et al. [DMCN12] sind bei Javier Diaz et al. zu finden. In diesem Paper wurden auch aktuelle Programmiermodelle untersucht. Das Ziel des Papers ist es, die Tauglichkeit der Modelle für die HPC

Community zu überprüfen. Dafür wurden die Programmiermodelle so unterteilt, dass die Modelle mit ähnlicher Zielarchitektur (Fokus auf Speichermodelle) zusammenkamen. Das heißt, eine große Gruppe sind die Shared- und Distributed Memory Models. Daneben kommen die GPGPU Models. Die restlichen Modelle, die nicht zu diesen beiden gehören wurden danach untersucht. Somit ergab sich noch dazu eine Gruppe für die Hybrid Models und abseits davon noch als einzelner Punkt das PGAS Modell. Diese Unterteilung war somit nützlich, da sie sehr grob ist. Die Autoren haben aus jeder Kategorie nur eine Auswahl an Programmiermodellen untersucht.

In dem Paper von Matevz Jekovec [Jek11] ist es das Ziel, die wichtigsten Models of Computation vorzustellen. Dazu gehören sequentielle Programmiermodelle mit den jeweiligen Gegenstücken auf der parallelen Seite. Für die Programmiermodelle wurden also die wichtigsten aus einer Generation ausgewählt. Die Generationen sind wieder Shared-, Distributed- und Hierarchical Memory Models. Die Klassifizierung zeigt auch an, welche Modelle eher abstrakt oder eher realistisch sind. In dem Paper heißt es, dass es wichtig ist technische Flaschenhälse zu untersuchen, um effiziente Algorithmen zu entwickeln. Der Autor weist auf die Speicherarchitektur als wichtiger Punkt hin. Hierarchical Memory Models liefern einen guten Ansatz, um den Flaschenhals zu umgehen. Die Unterteilung nach den Speichermodellen hilft somit einerseits dem Ziel wichtige Programmiermodelle aus dem sequentiellen Bereich mit ihren parallelen Gegenstücken vorzustellen, denn der Leser sieht somit die Ähnlichkeiten dieser Paare in der visualisierten Klassifizierung und die Unterschiede zu anderen Generationen. Andererseits werden jedoch auch Aspekte für die Entwicklung von Algorithmen angesprochen und wichtige Punkte dafür sind somit auch in der Klassifizierung enthalten. Gründe für die Ordnung nach Alter und abstrakten bis realistischen Modellen wurden nicht gefunden.

Evgenij Belikov et al. [BDT+13] verfolgt mit seinem Paper das Ziel effiziente und flexible Programmiermodelle mit High-Level Ansätzen zu untersuchen und diese vorzustellen. Die Autoren sehen in den traditionellen Programmiermodellen einige Nachteile gegenüber den High-Level Parallele Programming Models. Durch diesen Fokus auf die Effizienz der Programmiermodelle und den Hardwareproblemen ist auch die Klassifizierung dieser Modelle nach den Mechanismen (Behandlung von Coordination und Computation) und groben Gruppen (data-parallel, object-oriented, etc.) geordnet. High-Level Models müssen laut den Autoren eine Balance in der Abstraktion von Computation und Coordination finden. Um zu zeigen, wie die einzelnen Modelle dies handhaben, sind diese in der Klassifizierung so geordnet. Da in dem Paper Programmiersprachen und Bibliotheken in der Klassifizierung eingeordnet wurden, sind die Cluster auch auf Eigenschaften von Sprachen ausgerichtet, wie z.b. deklarativ oder imperativ.

Unterschiedliche Ziele in den Papern führen also zu unterschiedlichen Klassifizierungen. Zeitlich liegen die Paper nicht sehr weit auseinander (2011-2013). Eine Ausnahme bildet nur das Paper [ZCSM07] von 2007. Ein weiterer Unterschied ist auch, dass manche Paper sich auf Bibliotheken, wie OpenMP oder Sprachen, wie Fortress und Haskell konzentrieren. Andere gehen mehr auf die Modelle und Techniken, wie PRAM, PCO oder PMH ein. Wurden Sprachen und Bibliotheken untersucht, findet

sich auch die Gruppe der GPGPU Models wieder. Dies entweder als eigene Unterteilung oder eingeordnet in eine andere Gruppe, wie bei [Li11]. Wurden wiederum nur Techniken und Modelle untersucht, so findet sich z.B. bei Zhang Yunquan et al. kein Hinweis auf GPGPU Models und bei Matevz Jekovec nur eine kleine Erwähnung, da das K-Model bei den Hybriden Modellen zu den GPGPU Models gehört.

4. Erstellung einer eigenen Taxonomie

Nach dem nun die existierenden Taxonomien analysiert wurden, sollte eine Kurzsuche ausgeführt werden (siehe Einleitung 1.2). Diese Kurzsuche sollte zeigen, welche Abhängigkeiten die Programmiermodelle an die Hardware haben. Da in den ausgearbeiteten Quellen auf diese Frage eingegangen wurde, muss nicht mehr danach gesucht werden. In den Quellen wurden unter anderem mehrfach die Programmiermodelle auf Basis der Systemarchitektur (Speichermodelle) unterschieden. Diese Informationen können somit gleich bei der Erstellung der eigenen Taxonomie benutzt werden. Dies soll in diesem Kapitel passieren. Dabei sind zwei Fragen wichtig:

- Welches Ziel oder welche Motivation soll diese Taxonomie haben?
- Wie baut sich die Taxonomie auf und auf welcher Grundlage wurden Designentscheidungen getroffen?

4.1. Ziel der Taxonomie

Die Taxonomie soll, wie in der Motivation der Arbeit (Abschnitt 1.2, S. 12) schon beschrieben, als Entscheidungshilfe dienen. Sie soll helfen, herauszufinden, welche Unterschiede zwischen den einzelnen Programmiermodellen herrschen, besonders hinsichtlich der Hardwareanforderungen (geteilter oder gemeinsamer Speicher) und Kommunikationsmodellen. Die anfangs gestellte Forschungsfrage: *Wie können die Programmiermodelle der parallelen Programmierung einheitlich strukturiert werden und welche Unterscheidungsmerkmale besitzen sie?* wird betrachtet und im Folgenden beantwortet.

4.2. Grundlage und Aufbau der Taxonomie

Als Ausgangspunkt wurden mehrere Aspekte von den, im vorigen Kapitel, untersuchten Taxonomien genommen. Es zeigte sich, dass dort eine Unterteilung der Programmiermodelle aufgrund der benutzten Speichermodelle vorherrschend ist. D.h. es gibt Programmiermodelle mit Shared-, Distributed- oder Hierarchical Memory Models ([Jek11], [DMCN12], [ZCSM07]). Daneben taucht immer wieder die Gruppe der GPGPU Modelle auf ([BDT+13], [Li11], [DMCN12]) und dazwischen kann die Gruppe der hybriden Programmiermodelle eingeordnet werden, welche verschiedene Modelle aus den eben genannten Gruppen vereint. Das häufige Auftreten der GPGPU Modelle führt somit dazu, dass sie eine Gruppe in der Taxonomie erhielten. Sie werden weiterhin wahrscheinlich auch immer beliebter für viele Nutzer. Der Grund

für die Grupper der hybriden Modelle war folgender. Das Paper von Javier Diaz et al. [DMCN12] zeigte viele Möglichkeiten zur Kombination von verschiedenen Programmiermodellen auf. Einige Kombinationen fanden nach den Autoren auch schon Anwendungen in Systemen. Durch die Einbeziehung dieser kombinierten Modelle werden dem Nutzer der Taxonomie somit noch weitere Möglichkeiten eröffnet. Er könnte für spezielle Anwendungen versuchen die Vorteile zweier Programmiermodelle zu kombinieren.

Die Programmiermodelle, die sich in die einzelnen Gruppen befinden, sollen ebenso geordnet werden. Grund dafür ist, dass es unterschiedliche Formen gibt. Das heißt z.B. Bibliotheken OpenMP, MPI und IntelTBB gegenüber den Modellen PRAM, BSP und PGAS. Weiterhin sollen Informationen über jedes Modell vorhanden sein. Ein Unterscheidungsmerkmal ist z.B. auch die Data- oder Task Parallelität.

Der genaue Aufbau ist dargestellt in der folgenden Abbildung 4.1. Die Unterteilung der klassischen Modelle in den verschiedenen Speichermodellen bedeutet nicht, dass die zwei anderen Gruppen auf keine Speichermodelle aufbauen! CUDA und OpenCL basieren z.B. auf Hierarchical Memory [BDT+13]. Es ist auch nicht gleichzusetzen mit den erwähnten zeitlichen Generationen aus dem Paper [ZCSM07]. Deswegen wird MPI, dass mit dem Message Passing Modell auf Shared- und Distributed Memory aufbauen kann [DMCN12], nicht nur der Shared Memory Gruppe zugeordnet.

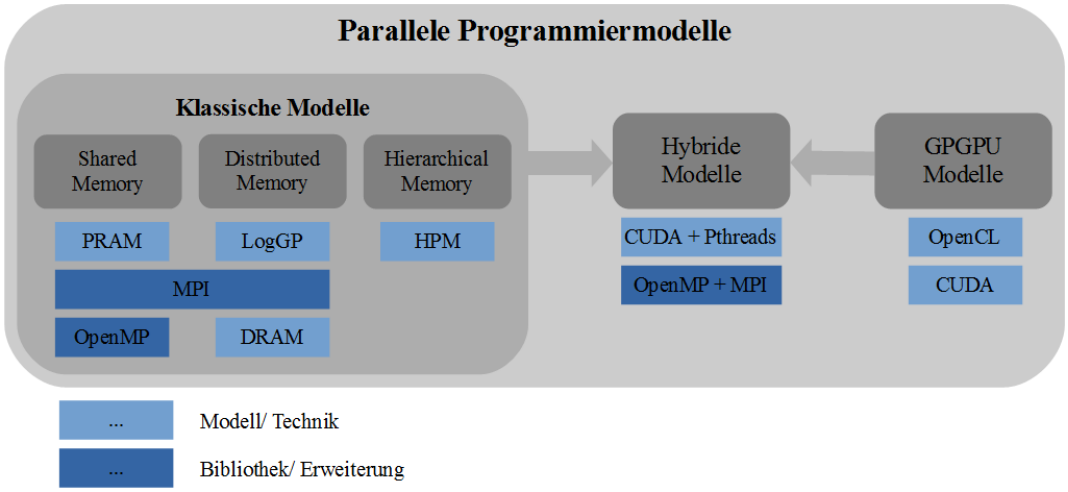

Abbildung 4.1.: Die Taxonomie als Ergebnis dieser Arbeit. Mehrere Elemente aus untersuchten Taxonomien der gefundenen Quellen bildeten die Grundlage. Die einzelnen Beispiele wurden nach den Informationen der Paper [BDT+13], [ZCSM07] und [DMCN12] eingeordnet.

4.3. Begriffe

- Parallel Programming Model bezeichnet Modelle und Techniken, wie PRAM sowie auch APIs, wie OpenMP.

- Memory Models bezeichnet das grundlegende Speichermodell. Das kann Shared, Distributed, Hierarchical, etc. sein.

4.4. Unterscheidungsmerkmale

- Klassische Modelle – Hybride Modelle – GPGPU Modelle
- Speichermodell (Shared, Distributed, Hierarchical)
- Data- oder Task parallel
- Typ (Erweiterung, Modell, ...)
- 7 Kriterien von [KMZS08] möglich (siehe 4.6)

4.5. Nutzen der Taxonomie

Diese Taxonomie liefert einen guten Überblick über die Programmiermodelle. Der Nutzer kann sich entweder in den Bereich der GPGPU Modelle begeben oder sich ein klassisches Modell aussuchen. Die Modelle sind nach dem jeweiligen Speichermodell geordnet und liefern Informationen über die Art des Modells als Abgrenzung zu den anderen Modellen in der Gruppe. So kann der Nutzer in der jeweiligen Kategorie sich das beste Modell passend zu seiner vorliegenden Hardware suchen. Es existiert auch eine Spalte für die Sprachen, die die Modelle implementieren könnten. So werden dem Nutzer Möglichkeiten zur Implementation vorgeschlagen.

4.6. Benutzung der Übersicht

In der Tabelle 4.1 ist ein Grundgerüst für eine Übersicht dargestellt, wie sie aus der Taxonomie abgeleitet werden kann. Die Taxonomie aus der Abbildung 4.1 gibt eine Unterscheidung der Programmiermodelle in Gruppen vor. In diesen Gruppen können die Programmiermodelle jedoch auch noch einmal unterschieden werden. Um die Problemsituationen aus dem Einleitungskapitel 1.2 zu beantworten, wurden die wichtigsten Spalten in die Übersicht eingefügt. Dazu gehören z.B. die Anforderungen an die Hardware zwecks geteiltem oder gemeinsamen Speicher. Der Benutzer kann nun aufbauend auf eine gewünschte Systemarchitektur eine Gruppe aus der Taxonomie wählen. In dieser Gruppe kann er durch die Unterscheidungskriterien ein Modell auswählen, dass seinen Ansprüchen entspricht. Es ist möglich weitere Spalten dieser Tabelle hinzuzufügen. Dies verringert die Übersichtlichkeit, gibt aber mehr Informationen über die Programmiermodelle preis. Wenn der Benutzer für seine Wahl genaue Unterscheidungen braucht, um die Performance seines Systems optimal zu halten, dann können die 7 Kriterien von [KMZS08] als Spalten hinzugefügt werden. Die Kriterien sind: 1.*system architecture*, 2.*programming methodologies*, 3. *worker management*, 4. *workload partitioning scheme*, 5. *task-to-worker mapping*, 6. *synchronization* und 7. *communication model* (alle Kriterien nachzulesen, bei der Auswertung

dieser Quelle im Unterabschnitt 2.3.1, S. 26). Hat der Benutzer sein Modell gewählt, hat er auch die Möglichkeit in der Gruppe der hybriden Modelle Informationen über Kombinationen zu holen. So kann er z.B. durch eine Kombination zweier Modelle für ein spezielles System bessere Performance erreichen.

	Programmiermodell	Typ	Nachrichtenaustausch	Data-/Task Parallel	Implementationen
Klassisch	PRAM	Modell	SM	Data	OpenMP, CUDA
	ArBB	Bibliothek	SM	Data	k.A.
	...				
	BSP	Modell	DM	k.A.	BSPlib
	LogP	Modell	DM	k.A.	k.A.
	...				
	P-HMM	Modell	HM	k.A.	k.A.
	...				
Hybrid	Pthreads&MPI	Bibliotheken	SM	Task/Data	MPICH2
	...				
	...				
GPGPU	OpenCL	Bibliothek	HM	Data	OpenCL C
	...				
	...				

Tabelle 4.1.: Grundform der Übersicht aus der erstellten Taxonomie. Die Informationen kommen aus den Quellen: [ZCSM07], [BDT+13], [Wri06] und [Wik16]. Nicht für alle Spalten wurden in diesen Quellen Informationen gefunden. SM, DM und HM bedeuten Shared-, Distributed- und Hierarchical Memory.

5. Zusammenfassung und Ausblick

In der Einleitung wurde darauf hingewiesen, dass durch das Fehlen einer einheitlichen Taxonomie unterschiedliche Probleme auftreten. Um eine einheitliche Taxonomie zu erstellen, wurde vorgeschlagen ein Systematic Literatur Review (SLR) durchzuführen. Dieses SLR wurde erfolgreich durchgeführt. Dabei wurde nach allen relevanten Surveys über die Programmiermodelle der parallelen Programmierung gesucht. Aus den insgesamt 20 gefundenen Quellen wurden durch diverse Ausschlusskriterien im Kapitel 2 genau 9 Quellen nicht in die Betrachtung aufgenommen. Die restlichen 11 Quellen wurden systematisch ausgewertet und lieferten viele wertvolle Informationen für diese Arbeit. Dazu wurden die Quellen nach den behandelten Programmiermodellen untersucht und daraus eine Übersicht erstellt. Dazu gehört eine Tabelle und eine strukturierte Übersicht über die Modelle mit kurzen Erklärungen. Anschließend wurden die Quellen einzeln durchsucht und viele Informationen über die angestellten Untersuchungen und den jeweils vorgestellten Taxonomie herausgeschrieben. So konnten Gemeinsamkeiten und Unterschiede ausgemacht werden. Es wurde somit die Annahme bestätigt, dass die derzeit existierenden Taxonomien unterschiedlich aufgebaut sind.

Im nächsten Schritt wurden die Taxonomien der Quellen einzeln untersucht. Keine der gefunden Taxonomien war gleich einer anderen. Es wurde herausgefunden, dass jede der Quellen ein anderes Ziel mit der Taxonomie erreichen wollte. Einige Elemente aus einer Taxonomie wurden auch in anderen Taxonomien gefunden. Dort wurde dies jedoch noch erweitert durch zusätzliche Unterscheidungen zwischen den Programmiermodellen. Konkret war das gemeinsame Element oft die Unterteilung der Programmiermodelle aufgrund ihrer Shared-, Distributed- oder Hierarchical Memory Models. Diese Unterteilung fand sich auch in anderen Taxonomien vor. Es konnten also neben den großen Unterschieden auch immer wieder Gemeinsamkeiten gefunden werden. Wichtig war deshalb die Unterschiede der Taxonomien auf die unterschiedlichen Ziele der Paper zurückzuführen, um herauszufinden welche Elemente für das Ziel dieser Arbeit wichtig sind und welche Elemente somit in der eigenen Taxonomie auftauchen müssen. Eine anfangs anschließend geplante Kurzsuche nach den Abhängigkeiten zur Hardware wurde übersprungen, da die untersuchten Quellen viele Informationen darüber gaben. Diese Informationen konnten dann benutzt werden.

Im anschließendem Kapitel wurde eine eigene Taxonomie erstellt. Diese baut auf viele Erkenntnisse aus den untersuchten Taxonomien auf und erweitert diese. Es wurde versucht eine Taxonomie nach einem vorher gestellten Ziel zu erstellen. In der Einleitung wurden diese Ziele/ Probleme genannt. Dazu gehört die Problemsituation, dass ein Softwareentwickler Informationen über die Abhängigkeiten der Modelle zu der Hardwarearchitektur braucht. In der Taxonomie werden die Modelle nach den

grundlegenden Speichermodellen aufgeteilt. Die Gruppe der GPGPU Models erhielt einen eigenen Platz. Zusätzliche Informationen sind zu Unterscheidung zu anderen Programmiermodellen der eigenen Gruppe vorhanden. So kann eine informierte Entscheidung getroffen werden.

Aus einer weiteren Situation kam die Frage nach den Konzepten der Implementierungen. Hierzu wurde gefunden, dass die Implementierungen auch nach den zugrunde legenden Speichermodellen unterteilt werden. Der Programmierer kann somit aus einer dieser Gruppen eine Sprache auswählen.

Dann stellte sich die Frage in der Einleitung, ob die vorhandenen Kurzübersichten ausreichend sind oder ob es notwendig ist Übersichten zu betrachten und zu erstellen, die alle Programmiermodelle enthalten. Einige der Quellen [ZCSM07] wiesen darauf hin, dass in Zukunft der Trend in Richtung Programmiermodelle mit Hierarchical Memory Models geht. Weiterhin wurde es herausgefunden , dass es Modelle gibt, wie MPI und OpenMP, die einen Standard in ihrer Gruppe darstellen. Es gibt also einen Trend in Richtung einer bestimmten Gruppe von Programmiermodellen und einige Modelle sind zum Standard für einige Programmierer geworden. Kurzübersichten sind somit ausreichend, wenn sie diesen Trend beachten und die derzeit wichtigsten Vertreter der Gruppen aufzeigen. Um gute Vergleiche zwischen Programmiermodellen zu erstellen, müssen jedoch ausführliche Übersichten aufgebaut werden. Solche Vergleiche sind immer wieder nötig, wenn die Performance eines Systems auf den höchsten Stand gebracht werden soll.

Die Fragen, die in der Einleitung gestellt wurden, sowie die Forschungsfrage konnten beantwortet werden. Um zu erfahren, ob de erstellte Taxonomie ausreichend ist, müsste sie eingesetzt und getestet werden. Weiterhin wurden in diesem Paper nicht alle existierenden Programmiermodelle gefunden. Es wurde sich auf die Programmiermodelle beschränkt, die in den Quellen gefunden wurden. Diese Quellen erschienen in dem Zeitraum von 2006-2016. Eine Übersicht über alle Programmiermodelle wurde angefangen. Im Anhang B wurden gefundene Programmiermodelle vorgestellt. Die Übersicht (Tabelle 4.1, S. 41) könnte anschließend mittels einer Studie über jedes einzelne Programmiermodelle mit Informationen gefüllt werden. Es wurden bestimmte Spalten dieser Tabelle vorgeschlagen, damit die Programmiermodelle mit wichtigen Informationen unterschieden sind. Es ist jedoch möglich diesen Vorschlag durch die 7 Kriterien von [KMZS08] zu erweitern, um somit noch mehr Unterscheidungskriterien zu erhalten. Vor- und Nachteile dazu wurden ausgewertet. Weiterführend könnte die Taxonomie auch erweitert werden, wenn neue Gruppen aufkommen. Die Übersicht muss dann gegebenenfalls mit mehr Informationen gefüllt werden.

Literaturverzeichnis

[Aga14] S. Agarwal; ***Models for parallel computing review and perspectives***.

[Agh85] G. A. Agha; ***Actors: A model of concurrent computation in distributed systems.***; Tech. report; DTIC Document; 1985.

[BDT+13] E. Belikov, P. Deligiannis, P. Totoo, M. Aljabri, & H.-W. Loidl; ***A survey of high-level parallel programming models***; Tech. report; Technical Report HW-MACS-TR-0103, Heriot-Watt University; 2013.

[BSP] ***Bsp - zusammenstellung damit verbundener paper***; `http://www.bsp-worldwide.org/implmnts/oxtool/papers.html`; [Online; Aufgerufen am: 24.08.2016].

[Cam94] D. K. Campbell; ***Clumps: a candidate model of efficient, general purpose parallel computation***; Ph.D. thesis; University of Exeter; 1994.

[Cil] ***Cilk einführung auf intel-webseite***; `http://www.cilk.com/`; [Online; Aufgerufen am: 24.10.2016].

[CUD] ***Cuda - offizielle webseite***; `http://www.nvidia.de/object/cuda-parallel-computing-de.html`; [Online; Aufgerufen am: 26.10.2016].

[DCH06] S. J. Deitz, B. L. Chamberlain, & M. B. Hribar; ***Chapel: Cascade high-productivity language an overview of the chapel parallel programming model***.

[DG08] J. Dean & S. Ghemawat; ***Mapreduce: simplified data processing on large clusters***; Communications of the ACM **51** (2008); Nr. 1; S. 107–113.

[DMCN12] J. Diaz, C. Munoz-Caro, & A. Nino; ***A survey of parallel programming models and tools in the multi and many-core era***; IEEE Transactions on parallel and distributed systems **23** (2012); Nr. 8; S. 1369–1386.

[Erl] ***Erlang - webseite***; `https://www.erlang.org/`; [Online; Aufgerufen am: 10.08.2016].

[Int] ***Intel arbb - archivinformation***; `https://software.intel.com/en-us/articles/intel-array-building-blocks`; [Online; Aufgerufen am: 11.11.2016].

[Jad] ***Jade - webseite***; `https://www.jadeworld.com/who-we-are/`; [Online; Aufgerufen am: 24.10.2016].

[JaJ11] J. F. JaJa; ***Pram (parallel random access machines)***; Springer US; Boston, MA; 2011.

[JAXC+13] L. Junzhi, Z. A-Xing, Q. Chengzhi, C. Lajiao, W. Hui, & J. Jingchao; ***Review on parallel computing of distributed hydrological models***; Progress in Geography **32** (2013); Nr. 4; S. 538–547.

[JB] M. Jekovec & A. Brodnik; ***Survey of the sequential and parallel models of computation: Technical report lusy-2012/02***.

[Jek11] M. Jekovec; ***Survey of the sequential and parallel models of computation***.

[JhED] J. C. Jenista, Y. hun Eom, & B. Demsky; ***Ooojava: Software out-of-order execution***.

[JR96] J. F. JaJa & K. W. Ryu; ***The block distributed memory model***; IEEE Transactions on Parallel and Distributed Systems **7** (1996); Nr. 8; S. 830–840.

[KC07] B. Kitchenham & S. Charters; ***Guidelines for performing systematic literature reviews in software engineering***; Technical report, Ver. 2.3 EBSE Technical Report. EBSE; 2007.

[KK07] C. Kessler & J. Keller; ***Models for parallel computing: Review and perspectives***; PARS Mitteilungen **24** (2007); Nr. 0177-0454; S. 13–29.

[KMZS08] H. Kasim, V. March, R. Zhang, & S. See; ***Survey on parallel programming model***; Network and Parallel Computing; Springer; 2008; S. 266–275.

[LCW+11] J.-J. Li, J. Cui, D. Wang, L. Yan, & Y.-S. Huang; ***Survey of mapreduce parallel programming model***; Dianzi Xuebao(Acta Electronica Sinica) **39** (2011); Nr. 11; S. 2635–2642.

[LHL+15] R. Li, H. Hu, H. Li, Y. Wu, & J. Yang; ***Mapreduce parallel programming model: A state-of-the-art survey***; International Journal of Parallel Programming (2015); S. 1–35.

[Li11] X. Li; ***A survey of task-based parallel programming models***; International Conference on Information Technology and Computer Science, 3rd (ITCS 2011); ASME Press; 2011.

[LT14] Y. Liu & G.-L. Tian; ***Sample size determination for the parallel model in a survey with sensitive questions***; Journal of the Korean Statistical Society **43** (2014); Nr. 2; S. 235–249.

[MHCS14] K. Mivule, B. Harvey, C. Cobb, & H. E. Sayed; ***A review of cuda, mapreduce, and pthreads parallel computing models***; arXiv preprint arXiv:1410.4453 (2014).

[MMT95] B. M. Maggs, L. R. Matheson, & R. E. Tarjan; ***Models of parallel computation: A survey and synthesis***; System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on; Bd. 2; IEEE; 1995; S. 61–70.

[MPI] ***Mpi - informationen***; `http://www-unix.mcs.anl.gov/mpi/`; [Online; Aufgerufen am: 26.10.2016].

[Opea] ***Opencl - webseite***; `https://www.khronos.org/opencl/`; [Online; Aufgerufen am: 24.10.2016].

[Opeb] ***Openmp - webseite***; `http://openmp.org/wp/`; [Online; Aufgerufen am: 10.08.2016].

[Owe08] J. Owens; ***Parallel programming models overview***; ACM SIGGRAPH 2008 classes; ACM; 2008; S. 13.

[PGA] ***Pgas - webseite***; `http://www.pgas.org/`; [Online; Aufgerufen am: 24.08.2016].

[Ram97] V. Ramachandran; ***Qsm: A general purpose shared-memory model for parallel computation***; S. 1–5; Springer Berlin Heidelberg; Berlin, Heidelberg; 1997.

[RGP12] B. RGPV; ***A review on new paradigm's of parallel programming models in high performance computing models in high performance computing***.

[SRA+11] C. Someswararao, K. B. Raju, S. Appaji, S. V. Raju, & K. Reddy; ***Recent advancements in parallel algorithms for string matching on computing models–a survey and experimental results***; International Conference on Advanced Computing, Networking and Security; Springer; 2011; S. 270–278.

[TBB] ***Intel thread building block - opensource website***; `https://www.threadingbuildingblocks.org/`; [Online; Aufgerufen am: 24.10.2016].

[Wik16] Wikipedia; ***Parallel programming model — wikipedia, the free encyclopedia***; `https://en.wikipedia.org/w/index.php?title=Parallel_programming_model&oldid=709885594`; 2016; [Online; Aufgerufen am: 30.05.2016].

[Wri06] C. Wright; ***Hybrid programming fun: Making bzip2 parallel with mpich2 & pthreads on the cray xd1***; CUG'06, 48th Cray User Group Conference; Bd. 1; Citeseer; 2006.

[ZCSM07] Y. Zhang, G. Chen, G. Sun, & Q. Miao; ***Models of parallel computation: a survey and classification***; Frontiers of Computer Science in China **1** (2007); Nr. 2; S. 156–165.

Anhang

A. SLR Tabelle

Titel des Papers	Ausgeschl.	Quelle
A Survey of High-Level Parallel Programming Models	Nein	[BDT+13]
Models for Parallel Computing: Review and Perspectives	Nein	[KK07]
Models for Parallel Computing Review and Perspectives	Nein	[Aga14]
Models of parallel computation: a survey and classification	Nein	[ZCSM07]
Survey of the sequential and parallel models of computation	Nein	[Jek11]
Survey on Parallel Programming Model	Nein	[KMZS08]
A Review of CUDA, MapReduce, and Pthreads Parallel Computing Models	Nein	[MHCS14]
A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era	Nein	[DMCN12]
A Review on New Paradigm's of Parallel Programming Models in High Performance Computing	Nein	[RGP12]
Recent Advancements in Parallel Algorithms for String Matching on Computing Models – A Survey and Experimental Results	Nein	[SRA+11]
A Survey of Task-Based Parallel Programming Models	Nein	[Li11]
MapReduce Parallel Programming Model: A State-of-the-Art Survey	Ja (E_6)	[LHL+15]
Chapel: Cascade High-Productivity Language An Overview of the Chapel Parallel Programming Model	Ja (E_6)	[DCH06]
Survey of MapReduce parallel programming model	Ja (E_4)	[LCW+11]
Sample size determination for the parallel model in a survey with sensitive questions	Ja (E_5)	[LT14]
Parallel programming models overview	Ja (E_3)	[Owe08]
Review on parallel computing of distributed hydrological models	Ja (E_2)	[JAXC+13]

Tabelle A.1.: Übersicht über alle durch das SLR gefundenen Quellen sowie die Beurteilung, ob sie ausgeschlossen (+ dazugehöriges Kriterium) oder betrachtet wurden.

B. Existierende parallele Programmiermodelle, Sprachen und Bibliotheken

In diesem Anhangs-Kapitel werden alle parallelen Programmiersprachen, -Modelle und Bibliotheken aus den Quellen aufgelistet und kurz beschrieben. Informationen über mögliche Implementierungen wurden aus den Quellen [KK07] und [Wik16] gewonnen.

B.1. Programmiermodelle

Alle Programmiermodelle nach der Definition dieser Arbeit werden im Folgenden aufgelistet.

PRAM

Parallel Random Access Machine (PRAM) ist ein paralleles Programmiermodell von 1978. Es basiert auf Shared Memory und Lock-Step Synchronisation. Es gehört somit zu der ersten Generation von Programmiermodellen [ZCSM07].
Implementierungen: Cilk, CUDA, OpenMP, TBB

BSP

Das Bulk Synchronous Parallel Model (BSP) [BSP] von 1990 basiert auf Distributed Memory. Laut dem Paper [ZCSM07] gehört es somit zu der zweiten Generation von Programmiermodellen. Die Synchronisation ist, wie der Name schon sagt, bulk-synchronous synchronisation.
Implementierungen: BSPlib, PUB

PGAS

Partitioned Global Address Space (PGAS) [PGA] ist ein Programmiermodell, das einen Global Address Space zusammen mit einem SPMD Model zur Verfügung stellt. Zu den Sprachen, die PGAS unterstützen gehören unter anderem Unified Parallel C und Fortran [DMCN12].
Implementierungen: NestStep, UPC, Fortran 2008

BDM

Block Distributed Memory Model (BDM) [JR96] ist ein Programmiermodell. Es entstand 1996 und basiert auf Distributed/Shared Memory und asynchroner Synchronisation. Es gehört zur zweiten Generation der Programmiermodelle [ZCSM07].

CLUMPS

Campbell's Lenient, Unified Model of Parallel Systems (CLUMPS) [Cam94] ist ein Programmiermodell von 1994. Es benutzt asynchrone Synchronisation und Distributed Memory. Es gehört somit zur zweiten Generation von Programmiermodellen [ZCSM07].

CUDA

CUDA [CUD] ist ein Programmiermodell und wurde von NVIDIA entwickelt. Dieses Projekt stammt von 2006. CUDA gehört zu den Heterogeneous Programming Models und ermöglicht die Abarbeitung von Tasks mit Hilfe der GPU [DMCN12].
Implementiert: PRAM

DirectCompute

DirectCompute gehört zu den Heterogeneous Programming Models. Es wurde von Microsoft für die GPU Programmierung entwickelt. Es funktioniert nur auf Windows-Plattformen [DMCN12].

HPM

Hierarchical Model for Parallel Computations (HPM) ist ein Programmiermodell, dass auf Distributed/Hierarchical Memory basiert. Es gehört laut dem Paper [ZCSM07] dadurch zur dritten Generation von Programmiermodellen.

LogP

Das LogP Modell besteht aus vier Parametern: L (Latency), o (overhead), g (gap) und P (number of computers). Das Modell von 1993 basiert auf Distributed Memory und gehört zur zweiten Generation von Programmiermodellen [ZCSM07].

NHBL

Non-dedicated Heterogeneous Barrier LogGP (NHBL) ist ein Programmiermodell, dass LogGP erweitert. Es entstand 2001 und basiert auf Distributed Memory. Es gehört somit zu der zweiten Generation von Programmiermodellen [ZCSM07].

OoOJava

OoOJava (Out-of-Order Java) ist ein paralleles Programmiermodell, dass Java erweitert. Wegen dieser Ähnlichkeit zu Java ist dieses relativ neue Modell sehr beliebt unter Programmierern, laut [JhED]. Es gehört zu den task-based data-driven Programming Models [Li11].

P-HMM & P-BT

Hierarchical Memory Model (HMM) und Hierarchical Memory Model mit Block Transfer (HMM&BT) sind zwei Programmiermodelle. Die parallelen Versionen P-HMM und P-BT sind 1993 entstanden. Diese zwei Modelle basieren auf Distributed/Hierarchical Memory und gehören zur dritten Generation der Programmiermodelle [ZCSM07].

Pthreads

Portable Operating System Interface (POSIX) Threads ist ein Programmiermodell, das als Header oder Bibliothek eingesetzt werden kann. In dem Paper [DMCN12] wird dieses Modell wegen mehrerer Gründe nicht für die Entwicklung von HPC (High-Performance Computing) Anwendungen empfohlen. Es baut auf Shared Memory auf.

SMPSs

SMP-Superscalar model ist ein Programmiermodell, dass die Parallelisierung automatisch berechnen und verwalten kann. Es analysiert die Abhängigkeiten zwischen Tasks und kann somit Informationen zur Parallelisierung sammeln. Es gehört zu den task-based data-driven Parallel Programming Models [Li11].

QSM

Queuing Shared Memory (QSM) [Ram97] ist ein Programmiermodell von 1997. Wie der Name verrät, baut es auf Shared Memory auf. Die Synchronisation ist bulk-synchronous und es gehört zu der ersten Generation von Modellen nach diesem Paper [ZCSM07].

MapReduce

MapReduce ist ein Programmiermodell von Google [DG08]. Es eignet sich zur Berechnung von großen Datensätzen. Der Name ergibt sich aus den zwei Funktionen map und reduce [MHCS14].

B.2. Bibliotheken

MPI

Message Passing Interface [MPI] ist eine Bibliothek aufbauend auf einer Shared und Distributed Memory Architektur und dem Message Passing Model. Es spezifiziert Syntax und Semantik für C, C++ und Fortran. Laut dem Paper [DMCN12] ist MPI der Standard für HPC Anwendungen geworden.
Implementierungen: OpenMPI, MPICH, MPI.NET

Intel TBB

Intel Threading Building Blocks [TBB] ist eine C++-Bibliothek. IntelTBB ist ein OpenSource Projekt seit 2006. Die Template-Bibliothek ist in dem Paper von XinLi [Li11] als task-based control-driven Programming Model aufgeführt.
Implementiert: PRAM

Intel ArBB

Intel Array Building Blocks war eine C++-Bibliothek von Intel. Das Projekt wurde aufgegeben [Int]. Gemäß dem Paper [DMCN12] ist ArBB eine Bibliothek zur Lösung von Daten intensiven Problemen mit Vektoren.

OpenMP

Mit Open Multi-Processing (OpenMP) Parallelisierung mit den Sprachen C++ und Fortress erreicht werden [Opeb]. OpenMP ist eine API (Schnittstelle) von 1997. Laut dem Paper von Xin Li [Li11] enthält OpenMP das SPMD Model und gehört zu den task-based Programming Models.
Implementiert: PRAM

OpenCL

Open Computing Language (OpenCL) [Opea] ist eine API mit der Parallelisierung über CPU und GPU erreicht werden kann. Laut Xin Li [Li11] kam OpenCL 1.1 2010 auf den Markt und gehört zu den task-based control-driven Parallel Programming Models.
Implementierungen: OpenCL C

B.3. Parallele Programmiersprachen

Cilk

Cilk [Cil] ist eine imperative Programmiersprache. Sie baut auf C und C++ auf und wurde von Intel entwickelt. Sie reicht zurück bis 1994. Cilk gehört nach dem Paper [Li11] zu den task-based control-driven Parallel Languages.

Implementiert: PRAM

Jade

Jade [Jad] ist eine Programmiersprache von 1993 aufbauend auf C. In dem Paper [Li11] schreibt Xin Li, dass Jade für jeden Task eine Data-Access Methode benötigt. Diese Methode muss vom Programmierer geschrieben werden. Deswegen gehört Jade zu den task-based data-driven Parallel Programming Models.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, den 14.11.2016

Markus Nestmann
338172